Universidad
Carlos III de Madrid

Implementación de una librería para el protocolo SIMPLE en J2ME

PROYECTO FIN DE CARRERA

Leganés, Agosto 2010

Autor: Daniel Conde García

Tutor: Mª Celeste Campo Vázquez

Título: Implementación de una librería para el protocolo SIMPLE en J2ME

Autor: Daniel Conde García

Director:

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Se lo agradezco a Adrián, mi mejor amigo y compañero que siempre ha estado ahí.

A mi familia por sus enseñanzas, su apoyo y paciencia durante estos años.

A mi tutora por darme la oportunidad de terminar con este ciclo.

Resumen

Actualmente la utilización de dispositivos móviles se ha convertido en un aspecto indispensable para nuestra sociedad. La necesidad de una constante disponibilidad hace patente el desarrollo de aplicaciones de comunicación entre los usuarios; la utilización de estándares en el desarrollo de estas aplicaciones permite la interoperabilidad de las mismas, por ello el desarrollo de librerías que definan estos estándares impulsa enormemente la creación de este tipo de aplicaciones lo que supone un beneficio directo para el desarrollo y el impulso de nuestra sociedad.

En el caso que nos ocupa se contempla el desarrollo de clientes de mensajería instantánea, un sistema que destaca por tener una respuesta más rápida que el correo electrónico, que permite efectuar multitarea al no ser necesaria una respuesta tan inmediata como la que se realiza durante una conversación telefónica y que nos informa del estado de la disponibilidad en el que un usuario se encuentra.

El objetivo de este proyecto es el de estudiar e implementar una librería utilizando las especificaciones Java JSR 164 y JSR 165 que definen el estándar de comunicación *Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions* (SIMPLE) dotando de esta funcionalidad a los terminales móviles. Este estándar, todavía en definición, está basado en el protocolo *Session Initiation Protocol* (SIP) diseñado para el descubrimiento de agentes de usuario y el establecimiento de una sesión de comunicación que desean compartir; ampliándolo al ofrecer un sistema de notificación y un modelo de persistencia para determinar el estado de un usuario, así como un método de intercambio de mensajes entre los usuarios. En el desarrollo del proyecto se utilizará la tecnología *Java 2, Micro Edition* (J2ME) especialmente diseñada para plataformas móviles.

Palabras Clave: SIMPLE, SIP, J2ME, mensajería instantánea, presencia, móvil.

Abstract

Currently, the use of mobile devices has become very important to our society. There is a need of being available all the time and because of this its necessary to develop communication applications. The interoperability of these applications depends on the use of standards during its development. Building libraries based on these standards is a direct benefit for our society.

We decide to support IM clients because they are faster than email and at the same time it allows a slower answer than a phone call. On the other hand it notifies the availability of the user.

The goal of this dissertation is to study and build a library based on the JSR164 and JSR 165 specifications for the SIMPLE standard in the mobile environment. The standard is still in definition and it's based on the SIP standard which is design for the discovery of user agents and the establishment of a session. The SIMPLE protocol expands the behaviour of SIP with the ability of handling persistence and the exchange of messages. To develop this project we will use the J2ME platform designed for mobiles.

Keywords: SIMPLE, SIP, J2ME, instant messaging, presence, mobile.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1.

Introducción y objetivos

1.1. Introducción

Actualmente las comunicaciones en tiempo real están teniendo un gran auge en el mundo empresarial, esto evoluciona en la necesidad de crear un protocolo que unifique los mensajes en tiempo real con la capacidad de notificar la disponibilidad y la presencia de un usuario ofreciéndolo para un gran número de dispositivos. Con este objetivo se están desarrollando distintos estándares cerrados impulsados por empresas privadas como IBM y Microsoft y otros estándares abiertos impulsados por *Internet Engineering Task Force* (IETF).

Con el objetivo y la motivación de impulsar este tipo de tecnologías se pretende diseñar una librería *Java 2 Micro Edition* (J2ME) que implemente el protocolo *Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions* (SIMPLE), el cual se define como una extensión del protocolo *Session Initiation Protocol* (SIP) ambos desarrollados por IETF. El protocolo SIP se define con dos funcionalidades, por un lado aporta movilidad al realizar ubicación de usuarios y por otro define un procedimiento de gestión de sesiones. El protocolo SIP es extensible y esta característica es utilizada por el protocolo SIMPLE que añade la funcionalidad de mensajería instantánea y manejo de información presencial.

El protocolo SIMPLE todavía está en desarrollo por lo que se trabajará sobre un subconjunto de los *Request For Comments* (RFCs) que lo componen:

- **RFC 3265 - Specific Event Notification:** Define un procedimiento para el manejo de suscripciones a eventos y notificaciones de los mismos.
- **RFC 3428 – Extension For Instant Messaging:** Define el modo de intercambiar mensajes entre usuarios.
- **RFC 3856 - A Presence Event Package:** Define un conjunto de elementos de dominio y un protocolo de comunicación que maneja la información de presencia.
- **RFC 3863 - Presence Information Data Format:** Define un formato común de representación de información presencial.
- **RFC 3903 - Extension For Event State Publication:** Extiende el framework que se ha desarrollado para el manejo de información presencial añadiendo la publicación del estado a un elemento distribuidor.

Para la realización del proyecto se estudian las especificaciones *Java Specification Request* (JSR) definidos por la *Java Community Process* (JCP) estas especificaciones pueden estar destinados a una arquitectura concreta o ser genéricos para la comunidad Java. En nuestro caso se van a tratar con los siguientes JSR:

- **JSR 180 SIP API for J2ME:** este *application programming interface* (API) definido para realizar conexiones SIP en dispositivos limitados; se ha desarrollado bajo el liderazgo de Nokia y es específico del entorno J2ME.
- **JSR 164 SIMPLE *Presence* y JSR 165 SIMPLE *Instant Messaging*:** estas especificaciones son genéricas y no formar parte del entorno J2ME, en ellas se define el protocolo SIMPLE y están desarrolladas para que puedan coexistir con la implementación JAIN de SIP, con el JSR180 o con ninguna de las dos. Ambas especificaciones fueron liderados por el *Panasonic Information and Network Technologies Laboratory.*

Como los JSR 164 y JSR 165 no son específicos de J2ME no se incluyen como librerías de desarrollo para la arquitectura por lo que se encuentran pocas implementaciones de los mismos en los terminales, se han encontrado los terminales Sony Ericson TM506 y el Motorola KRZR K1 que implementan el JSR 165 para la mensajería instantánea pero no la parte del JSR 164 de presencia.

Así este proyecto consiste en el desarrollo de los JSR 164 SIMPLE *Presence* y JSR 165 SIMPLE *Instant Messaging* para el entorno J2ME utilizando como base la especificación JSR 180 SIP API.

1.2. Objetivos

Se pretende desarrollar una librería que implemente el estándar SIMPLE solucionando las necesidades citadas anteriormente al ofrecer una capa base desde la que desarrollar sistemas de mensajería instantánea con soporte para la notificación de la presencia de los usuarios.

Para ello se debe extender el protocolo SIP añadiendo la capacidad de mandar y recibir mensajes, realizar publicaciones del estado del usuario y permitir suscribirse y recibir notificaciones de otros; implementando los elementos necesarios para ello.

Los principales objetivos se listan a continuación:

- **Estudiar el dominio del proyecto:**

  Se deber realizar un estudio de los protocolos SIP y SIMPLE, así como de la arquitectura J2ME.

- **Desarrollar la librería SIMPLE:**

  Implementar un agente de usuario que soporte la comunicación SIP y que extienda y gestione los mensajes SIMPLE citados anteriormente.

- **Realizar un prototipo sobre la librería:**

  Implementar una aplicación J2ME que desarrolle un prototipo de chat, para mostrar la viabilidad de la librería, para la creación de este tipo de aplicaciones.

- **Evaluar ambos proyectos:**

  Se dispone de dos escenarios para la evaluación del proyecto, uno de simulación de terminales móviles y otro utilizando un terminal real. La evaluación debe recoger la siguiente funcionalidad:

  - Transmitir información presencial del usuario
  - Recibir información presencial de otros usuarios
  - Realizar una conversación de mensajería instantánea
    - Enviar mensajes
    - Recibir mensajes

- **Documentar el trabajo realizado:**

  Realizar el proceso de documentación asociado al proyecto que facilita su estudio y comprensión.

1.3. Fases del desarrollo

Para el desarrollo de la aplicación se ha utilizado la metodología ESA Lite (1). Esta metodología ha sido desarrollada por la Agencia Espacial Europea como estándar para el desarrollo de proyectos software y está orientada a proyectos de menor envergadura que el estándar original de la agencia espacial europea o *European Space Agency* (ESA), siendo los límites relativos definidos en ESA BSSC (1):

- Menos de dos años / persona de desarrollo.
- Un único equipo de desarrollo de 5 ó menos personas.
- La cantidad de código fuente producida debe ser inferior a 10.000 líneas de código, excluyendo comentarios.

Las ventajas de adoptar esta metodología orientada a proyectos pequeños en vez de una más pesada como ESA o Métrica 3 son (1):

- Combinar las fases de requisitos software y diseño de la arquitectura.
- Simplificar la documentación.
- Simplificar la planificación.
- Reducir los requisitos de confiabilidad.
- Utilizar la especificación de prueba del sistema para las pruebas de aceptación.

1.3.1. Identificación de las fases en el documento

Debido a que la estructura del presente documento sigue un patrón diferente al definido por ESA Lite, el cual se compone de varios documentos separados como se puede observar en las salidas de los procesos en la Figura 1, se indica la correspondencia entre los documentos de ESA Lite y los apartados del presente documento:

- **UR (REQUISITOS DE USUARIO):** Corresponde a los apartados 3.1.1, 3.1.2 y 0 para el desarrollo de la librería y para el desarrollo del prototipo corresponde a los apartados 4.1.1, 4.1.2, 4.1.3 y 4.1.4
- **SR/AD (REQUISITOS SOFTWARE / DISEÑO DE LA ARQUITECTURA):** Corresponde a los apartados 3.2.1 y 3.2.2 para el desarrollo de la librería y para el desarrollo del prototipo corresponde a los apartados 4.1.5, 4.1.6, 4.2.1 y 4.2.2.
- **DD (DISEÑO DETALLADO):** Corresponde a los apartados 3.2.3, 3.2.4 y 3.2.5 para el desarrollo de la librería y para el desarrollo de prototipo corresponde a los apartados 4.2.3 y 4.2.4.
- **TR (PRUEBAS DE VALIDACIÓN Y VERIFICACIÓN):** Corresponde al Capítulo 5 del documento.

- **OM (MANTENIMIENTO DEL SISTEMA):** Al no implantarse en el momento presente el sistema no se ha realizado un programa de mantenimiento. Este se trataría en caso de producirse esta implantación.

El ciclo de vida del desarrollo de una aplicación con ESA Lite se corresponde con el presentado en la Figura 1.

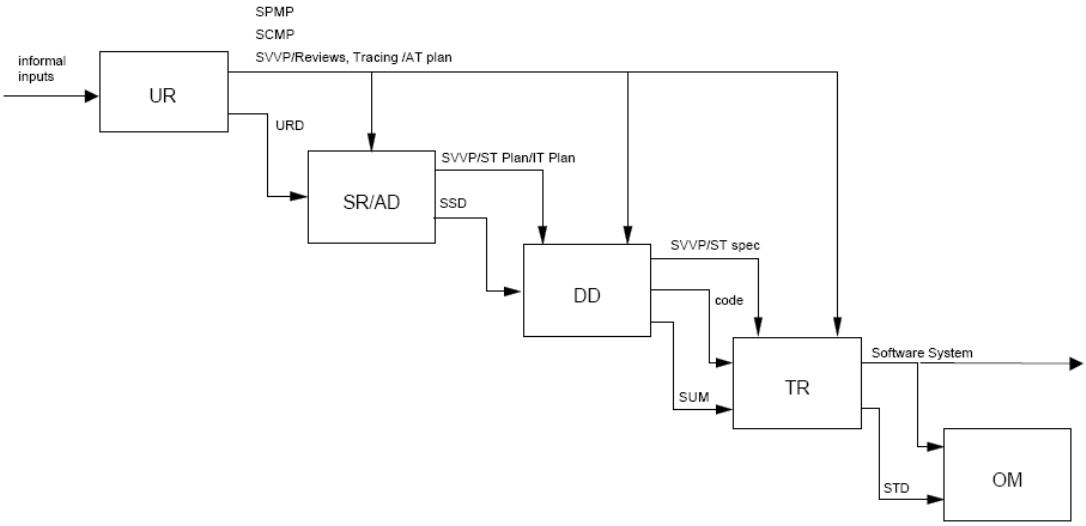


Figura 1: Ciclo de vida de un proyecto software con ESA Lite (1)

1.4. Medios empleados

Para la realización del proyecto se ha hecho uso de los siguientes recursos hardware y software:

1.4.1. Recursos hardware

Para la realización del software y la memoria se han utilizado 4 equipos informáticos con las siguientes características:

- Portátil Aspire 9413ZWXMi, Intel Pentium dual-core 1,73 GHz y 2 GB DDR2 de RAM (utilizado para la documentación)
- Portátil Apple MacBook 5.1, Intel Core 2 Duo 2.4 GHz, 4GB de RAM (utilizado para el diseño de diagramas y figuras)
- PC de desarrollo, AMD Athlon™ 64xDual 3800+, 2GHz y 2GB de RAM
- PC de ayuda en las pruebas, AMD Athlon™ XP1800+ y 2GB de RAM
- Nokia N80

1.4.2. Recursos software

El software detallado a continuación ha sido utilizado en al menos uno de los equipos anteriores:

Sistemas Operativos:

- Microsoft Windows Vista Home Premium: Sistema operativo del equipo portatil
- Mac OS X 10.5: Sistema operativo del equipo MacBook
- Microsoft Windows XP Profesional: Sistema operativo del equipo de desarrollo
- Ubuntu 10.04: Sistema operativo del equipo de ayuda en las pruebas

***Integrated Development Enviroment* (IDE) y plataforma:**

- NetBeans IDE 6.8
- Java(TM) Platform Micro Edition SDK 3.0

Software de pruebas y debug:

- Wireshark Versión 1.0.0
- Nokia PC Suite 6.80.21
- SIP Communicator 1.0-alpha6-nightly.build.2772 +

Software de documentación:

- ArgoUML Versión 0.30.1
- Microsoft Office 2007: Suite de ofimática
- OmniGraffle Professional: Software de creación de gráficos y diagramas
- Gimp: Software de diseño gráfico para la creación de los iconos y las imágenes de la aplicación como ayuda en el desarrollo de algunas figuras

Software de control de versiones:

- SVN Server: Servidor de control de versiones
- Dropbox: Software de sincronización de ficheros similar a SVN pero no específico para código fuente. Utilizado para la compartición del fichero de la memoria

1.5. Estructura de la memoria

A continuación se expone la estructura seguida en el presente documento, indicando el contenido de cada uno de los capítulos siguientes:

- **Capítulo 2 - Estado del arte:** se analizan las tecnologías actuales en el dominio tratado, realizando una breve descripción en las requeridas como conocimiento para el desarrollo completo del proyecto.
- **Capítulo 3 - Desarrollo de la librería SIMPLE:** se especifica el análisis y diseño de la librería, exponiendo el alcance del sistema y los estándares utilizados en el desarrollo, se especifica el diseño arquitectónico, las tecnologías utilizadas, las pautas de diseño y se realiza una descripción de las clases desarrolladas.
- **Capítulo 4 - Desarrollo del proyecto SChat:** se especifica el análisis y el diseño del prototipo, se incluyen los requisitos de usuario del mismo realizando una matriz de trazabilidad con los requisitos funcionales que generan. En cuanto al diseño se detallan los distintos diagramas de clases del sistema realizando una especificación de la funcionalidad.
- **Capítulo 5 - Pruebas:** se exponen las pruebas realizadas al prototipo y la librería con el objetivo de comprobar el cumplimiento de los requisitos establecidos en la fase de análisis.
- **Capítulo 6 - Historia del proyecto:** se presenta la planificación realizada para el proyecto así como un esbozo de los problemas más importantes encontrados durante la realización del mismo.
- **Capítulo 7 - Presupuesto:** se presenta la hoja desglosada con el coste de realización del proyecto incluyendo los gastos directos e indirectos del mismo así como el coste de personal.
- **Capítulo 8 - Conclusiones y trabajos futuros:** se exponen las conclusiones del autor a la realización del proyecto y se presentan un conjunto de posibles líneas futuras.

Capítulo 2.

Estado del arte

2.1. Introducción

Este capítulo ofrece una visión actual del panorama en el dominio de la mensajería instantánea ofreciendo una visión básica de las tecnologías utilizadas.

Primero se desarrolla la idea de en qué consiste la mensajería instantánea y su historia. Segundo se realizará un estudio de las tecnologías actuales, esto incluye tanto los estándares propuestos, como las implementaciones actuales de los mismos. Seguidamente se dará una breve introducción al protocolo SIP explicando en qué consiste y dando una idea general de su funcionamiento. Como tercer punto se realizará una introducción más extensa al protocolo SIMPLE, se desglosará el funcionamiento del protocolo indicando los distintos RFCs que intervienen en él y la función que aportan al protocolo. Después se dará una pequeña introducción a la tecnología J2ME que se utilizará para desarrollar la librería indicando las bondades que ofrece. Finalmente se explicará la funcionalidad que definen los estándares Java seleccionados para el desarrollo de la librería.

2.2. ¿Qué es la Mensajería Instantánea?

La mensajería instantánea engloba una serie de tecnologías utilizadas para la comunicación de texto en tiempo real entre dos o varios participantes, esta comunicación se realiza a través de alguna red ya sea privada o pública como puede ser Internet. Este tipo de mensajería se diferencia de otras tecnologías como pueda ser el e-mail, en la percepción que tienen los usuarios de estar realizando una comunicación síncrona.

La mensajería instantánea ofrece una manera eficiente y efectiva de comunicación permitiendo una inmediata respuesta o aceptación del mensaje, también ofrece una visión en tiempo real del estado actual de los contactos del usuario. La mensajería instantánea en muchas ocasiones se extiende con otros servicios populares como puede ser la transmisión de imágenes en forma de emoticonos, la transmisión de ficheros ligeros, la utilización de vídeo en tiempo real utilizando webcams o realización de llamadas utilizando tecnología de voz sobre IP (VoIP).

2.2.1. Características

Las principales características de la mensajería instantánea son los contactos y las conversaciones y sobre estas dos características se realizan la mayoría de los servicios de los distintos clientes.

Funcionalidad de los contactos:

- Disponer y gestionar una lista de contactos
- Aceptar y rechazar peticiones de subscripción de otros usuarios
- Mostrar varios estados, mensajes de estado, apodos y avatares

Funcionalidad de las conversaciones:

- Mandar distintos tipos de mensaje, pueden o no iniciar conversaciones
- Realizar conversaciones con múltiples usuarios
- Utilización de emoticonos

Otra funcionalidad común:

- Permitir distintos sistemas de comunicación, véase VoIP, pizarras electrónicas, etc.

2.2.2. Historia

La mensajería instantánea precede a Internet, aparece por primera vez en sistemas operativos multi-usuarios como CTSS y Multics(2) a mediados de la década de 1960. Con la aparición de las redes su desarrollo aumentó y aparecieron sistemas que utilizaban protocolos peer-to-peer (talk, ntalk e ytalk)(3) y sistemas cliente-servidor (talker e IRC)(4)(5). Con el apogeo de los sistemas de tablón de anuncios o B*ulletin Board Systems* (BBS) en la década de 1980 aparecen los primeros sistemas en utilizar el chat con las características similares a la mensajería instantánea (Freelancin' Roundtable).

En la última mitad de la década de 1980 y en la década de 1990 la compañía Quantum Link ofrece un servicio en línea para los ordenadores Commodore 64 que permite mandar mensajes entre usuarios conectados, a este servicio lo llamaron On-Line-Message (OLM); más tarde Quantum Link se convirtió en America Online creando AOL Instant Messenger (AIM) (6).

Con el apogeo de Internet a mediados de la década de 1990 aparecen los primeros clientes gráficos GUI de mensajería tal y como se conocen hoy en día (PowWow, ICQ (7) y el ya citado AOL Instant Messenger). Una funcionalidad similar ofrece CU-SeeMe en 1992 principalmente chat de audio y video pero ofrece también la posibilidad de enviar mensajes entre los usuarios. AOL adquiere Mirabilis creadores de ICQ, a los que la oficina de patentes de USA les concedería dos patentes para la mensajería instantánea. Mientras el resto de compañías desarrollan sus propias aplicaciones de mensajería instantánea (Excite, MSN, Ubique y Yahoo) cada una de ellas con su propio protocolo propietario. Ante esto los usuarios se ven obligados a utilizar múltiples clientes. En 1998 IBM lanza IBM Lotus Sametime (8) un producto basado en la tecnología adquirida a Ubique y a Databeam cuando las compró.

A principios de la década del 2000 aparecen los primeros RFCs dedicados al estándar SIMPLE(9), también aparece una aplicación libre Jabber basada en un protocolo de estándar abierto, el protocolo se estandarizó bajo el nombre de *eXtensible Messaging and Presence Protocol* (XMPP) (10) y permitía a los servidores actuar como pasarelas para otros protocolos de mensajería instantánea reduciendo así la necesidad de múltiples clientes.

Actualmente muchos servicios de mensajería instantánea están incluyendo otras funcionalidades como la video llamada, VoIP y las conferencias vía web, también se incluyen funcionalidades de compartición de escritorio e IP radio.

Por último el término *"Instant Messenger"* es una marca registrada de Time Warner y no puede utilizarse en software no afiliado a AOL dentro de Estados Unidos(11).

2.3. Estudio de las distintas tecnologías

Actualmente existe una gran diversidad de protocolos para realizar mensajería instantánea en la Tabla 1 se muestran los protocolos que se estudiarán.

Tabla 1: Protocolos de mensajería instantánea

Protocolo	Licencia	Creador	Publicación
IRC	Estándar libre	Jarkko Oirkairen	1988 Ago
MSNP	Propietario	Microsoft	1999 Jul
OSCAR	Propietario	AOL	1997
SIMPLE	Estándar libre	IETF	2002 Dic
Skype protocol	Propietario	Skype	-
TOC2	Propietario	AOL	2005 Sep
XMPP	Estándar libre	Jeremie Miller , estandarizado por IETF	1999 Ene
YMSG	Propietario	Yahoo!	-
Zephyr	Estándar libre	*Massachusets Institute of Technology* (MIT)	1987

A continuación se va a realizar una breve introducción de cada uno de los protocolos mencionados, para cada protocolo se ha creado una tabla de características[1] en la que se definen los siguientes campos:

- **Nombre del protocolo:** nombre completo y siglas si procede.
- **Identificador:** especifica el formato que se utiliza para identificar a los distintos usuarios dentro del protocolo.
- **Intercambio asíncrono:** Indica si la comunicación es asíncrona.
- ***Transport Layer Security* (TLS):** Indica si ofrece la opción de utilizar una capa de transporte segura.
- **Contactos ilimitados:** Indica si tiene límite en el número de contactos.
- **Mensajes a todos:** Indica si tiene un protocolo para intercambiar mensajes con todos los contactos.
- **Mensajes a muchos:** Indica si puede establecer grupos de conversación.
- **Protección SPAM:** Indica si el protocolo ofrece algún tipo de control contra SPAM.
- **Audio/VoIP:** Indica si el protocolo acepta el envío y recepción de datos de voz.
- **Video/Webcam:** Indica si el protocolo acepta el envío y recepción de datos de vídeo.

[1] Para indicar que no se ha encontrado información sobre un campo de la tabla se utilizará el carácter '-'.

2.3.1. Protocolo IRC

Internet Relay Chat (IRC) (12) (13) (14) (15) es un protocolo de intercambio de texto en tiempo real, principalmente está diseñado para comunicaciones de grupo en lo que llaman canales, pero también permite la comunicación personal mediante chats o con mensajes privados, el protocolo también permite la transferencia de datos utilizando *Direct Client to Client* (DCC).

Los servidores de IRC se conectan entre ellos para formar una red, los usuarios pueden acceder a estas redes conectando un cliente (mIRC o XChat) (16) (17) a alguno de los servidores. El protocolo utiliza *Transpor Control Protocol* (TCP) u opcionalmente TLS, la *Internet Assigned Numbers Authority* (IANA) asignó el puerto 194/TCP al protocolo pero para evitar ejecutarlo con privilegios de root se utiliza por defecto el puerto 6667/TCP y/o alguno cercano.

Tabla 2: Características del protocolo IRC

Nombre del protocolo	Internet Relay Chat (IRC)		
Identificador	Nickname!Username@hostname, ej. user!~usr@a.b.com		
Intercambio asíncrono	Si	**TLS**	Si
Contactos ilimitados	No	**Mensajes a todos**	No
Mensajes a muchos	Multicast simplificado	**Protección SPAM**	Media
Audio/VoIP	No	**Video/Webcam**	No

2.3.2. Protocolo MSNP

El protocolo MSNP (18) es un protocolo de mensajería instantánea desarrollado por Microsoft, el servicio acepta los clientes de mensajería Windows Live Messenger y es compatible con clientes de terceros como Pidgin (19) y Trillian (20).

El protocolo MSNP está diseñado para trabajar con el protocolo .Net Messenger Service, este último protocolo se encarga de la gestión de la presencia y el envío de mensajes entre las aplicaciones nativas de Microsoft.

Tabla 3: Características del protocolo MSNP

Nombre del protocolo	Windows Live Messenger (MSNP)		
Identificador	Dirección e-mail (Windows Live ID)		
Intercambio asíncrono	Si	**TLS**	No
Contactos ilimitados	Para robots certificados	**Mensajes a todos**	No
Mensajes a muchos	Centralizado	**Protección SPAM**	No
Audio/VoIP	-	**Video/Webcam**	-

2.3.3. Protocolo OSCAR

El protocolo *Open System form CommunicAtion in Realtime* (OSCAR) (21) es un protocolo para la mensajería instantánea que también maneja información de presencia, pertenece a AOL y actualmente se utiliza en los sistemas de mensajería ICQ y AIM.

A pesar del nombre la especificación del protocolo es propietaria, manteniendo a otros competidores incapaces de crear clientes compatibles. En 2002 Apple.Inc consiguió un contrato por el que se le permitía utilizar el protocolo en su aplicación iChat.

En marzo del 2008 AOL reveló porciones de la documentación de OSCAR, en 2009 grandes partes del protocolo han sido obtenidas mediante ingeniería inversa y han aparecido clientes de terceros que lo utilizan.

Tabla 4: Características del protocolo OSCAR

Nombre del protocolo	Open System for CommunicAtion in Realtime (OSCAR protocol)		
Identificador	Username, dirección e-mail o UIN, ej. 12345678		
Intercambio asíncrono	Si	**TLS**	Si(AIM pro, AIM Lite)
Contactos ilimitados	No	**Mensajes a todos**	No
Mensajes a muchos	Centralizado	**Protección SPAM**	Basado en el cliente
Audio/VoIP	Si	**Video/Webcam**	Si

2.3.4. Protocolo SIMPLE

SIMPLE es un protocolo de mensajería instantánea que maneja información de presencia, está basado en el protocolo SIP siendo una extensión de éste. El protocolo está mantenido por SIMLPE IETF Group (9).

La descripción del protocolo se detallará en un apartado posterior.

Tabla 5: Características del protocolo SIMPLE

Nombre del protocolo	Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions (SIMPLE)		
Identificador	user@hostname		
Intercambio asíncrono	Si	**TLS**	Si
Contactos ilimitados	Si	**Mensajes a todos**	Si
Mensajes a muchos	No	**Protección SPAM**	Media
Audio/VoIP	Si	**Video/Webcam**	Si

2.3.5. Protocolo de Skype

Skype (22) utiliza un protocolo privado propietario de VoIP basado en una arquitectura peer-to-peer. Esto hace que la red de Skype no sea compatible con la mayoría de las otras redes VoIP sin una licencia de Skype.

Al utilizar una arquitectura peer-to-peer se requiere una mínima infraestructura centralizada, el directorio de usuarios está distribuido entre los clientes y los nodos de la red. La red contiene 3 tipos de entidades: supernodos, nodos y servidores de autenticación. Cualquier cliente que no utilice firewall y con un buen acho de banda se transforma en un supernodo, transmitiendo así las comunicaciones a otros clientes que utilizan firewalls o *Network Address Translation* (NAT).

Lo poco que se sabe del protocolo es que las señales se encriptan utilizando RC4 y que la voz se encripta con AES.

Tabla 6: Características del protocolo de Skype

Nombre del protocolo	Skype protocol		
Identificador	Username		
Intercambio asíncrono	No	**TLS**	Propietario
Contactos ilimitados	-	**Mensajes a todos**	No
Mensajes a muchos	-	**Protección SPAM**	-
Audio/VoIP	Si	**Video/Webcam**	Si

2.3.6. Protocolo TOC2

El protocolo *Talk to OSCAR 2* (TOC2) (23) es una actualización del protocolo TOC. El protocolo está basado en ASCII y no tiene muchas de las características de OSCAR, entre ellas la de transmitir ficheros aunque sí puede recibirlos.

Tabla 7: Características del protocolo TOC2

Nombre del protocolo	Talk to OSCAR (TOC2)		
Identificador	Username o UIN, ej. 12345678		
Intercambio asíncrono	Si	**TLS**	No
Contactos ilimitados	No	**Mensajes a todos**	No
Mensajes a muchos	Centralizado	**Protección SPAM**	No
Audio/VoIP	-	**Video/Webcam**	-

2.3.7. Protocolo XMPP

XMPP (10) es un protocolo abierto, basado en *eXtensible Markup Language* (XML), diseñado para comunicaciones de mensajería instantánea y transferencia de información de presencia. Se desarrolló por la comunidad de Jabber en 1999, en el año 2002 el IETF formó un grupo de trabajo para XMPP para formalizar el núcleo del protocolo como una especificación IETF.

Ventajas:

- **Descentralización:** No se requiere de ningún servidor central
- **Estándares abiertos:** Con especificaciones en los RFC 3920 y RFC 3921
- **Historia:** lleva usándose desde 1998 y existen múltiples implementaciones
- **Seguridad:** apoya los sistemas de cifrado ofreciendo certificados digitales gratuitos
- **Flexibilidad:** permite realizar implementaciones a medida compatibles

Desventajas:

- **Sobrecarga de datos de presencia:** cerca del 70% del tráfico es de presencia.
- **Escalabilidad:** Sufre problemas de redundancia.
- **Sin datos binarios**: Se codifica como un documento XML

Tabla 8: Características del protocolo XMPP

Nombre del protocolo	Extensible Messaging and Presence Protocol (XMPP)		
Identificador	Jabber ID (JID) ej. usr@a.b.c/resource2		
Intercambio asíncrono	Si	**TLS**	Si
Contactos ilimitados	Si	**Mensajes a todos**	Si
Mensajes a muchos	Listas Unicast	**Protección SPAM**	Si
Audio/VoIP	Si	**Video/Webcam**	Si

2.3.8. Protocolo de Yahoo! Messenger

El protocolo *Yahoo! MeSsenGer protocol* (YMSG) (24) es un protocolo propietario de mensajería instantánea con manejo de información de presencia, usa conexiones TCP en el puerto 5050 y ofrece peticiones de *Hypertext Transfer Protocol* (HTTP) para clientes que estén tras un firewall. El protocolo permite transferencia de ficheros vía HTTP, servicio de webcam a través de H.323 y llamadas VoIP utilizando SIP.

Tabla 9: Características del protocolo YMSG

Nombre del protocolo	Yahoo! Messenger Protocol (YMSG)		
Identificador	Username		
Intercambio asíncrono	Si	**TLS**	No
Contactos ilimitados	No	**Mensajes a todos**	Si

[2] "Resource" distingue al mismo usuario permitiendo conectarse a la vez en varias localizaciones.

Mensajes a muchos	Centralizado	**Protección SPAM**	Si
Audio/VoIP	Chat rooms	**Video/Webcam**	Si

2.3.9. Protocolo Zephyr

Creado por el MIT, como parte del proyecto Athenea, se diseñó como un protocolo de mensajería instantánea para Unix. El proyecto se realizó utilizando distintos programas que trabajando juntos crean el sistema completo de mensajería Zephyr (25).

Zephyr usa datagramas *User Datagram Protocol* (UDP) y utiliza los puertos 2102, 2103 2104. Es incompatible con la mayoría de los routers que utilizan NAT. En la mayor parte de las implementaciones se utiliza Kerberos para la autenticación.

Tabla 10: Características del protocolo Zephyr

Nombre del protocolo	Zephyr Notification Service		
Identificador	Mediante Kerberos, ej. user@ATHENA.MIT.EDU		
Intercambio asíncrono	Si	**TLS**	No
Contactos ilimitados	Si	**Mensajes a todos**	Si
Mensajes a muchos	Listas unicast	**Protección SPAM**	No
Audio/VoIP	No	**Video/Webcam**	No

2.4. Introducción al protocolo SIP

2.4.1. ¿Qué es SIP?

SIP es un protocolo a nivel de aplicación definido por IETF en el RFC 3261 (26), se diseñó con el objetivo de la creación y el mantenimiento de sesiones sobre redes IP. Una sesión consiste en la definición de un canal de comunicación, tanto a nivel del tipo de medio (voz, video, datos) que se utilizará entre los participantes como de la codificación. Funciona sobre TCP y UDP en el puerto 5060 para comunicaciones no encriptadas y sobre el puerto 5061 en comunicaciones que utilizan TLS.

El protocolo SIP sigue una estructura similar al protocolo HTTP en cuanto a las transacciones de peticiones y respuestas, y un formato de texto similar en los mensajes como se muestra en la Figura 2.

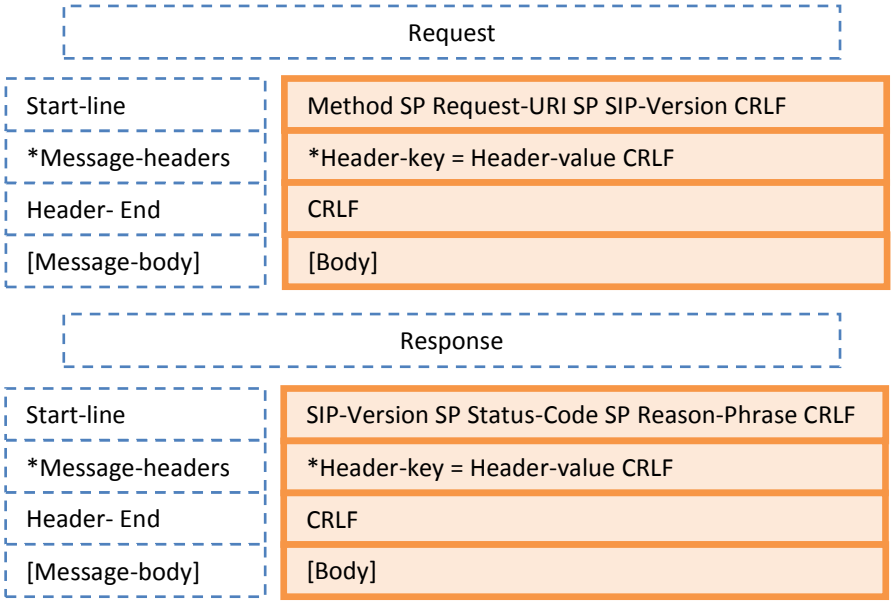


Figura 2: Formato de mensajes SIP

Las peticiones SIP se diferencian según el método que contienen:

- **REGISTER:** Indica la dirección IP y las URL de un agente de usuario.
- **INVITE:** Establece una sesión entre agentes de usuario.
- **ACK:** Confirma la recepción de mensajes.
- **CANCEL:** Termina una petición pendiente.
- **BYE:** Termina la sesión entre dos agentes de usuario
- **OPTIONS:** Realiza una petición de las capacidades del *caller*, sin aceptar la llamada.

Las respuestas SIP se diferencian según el código de respuesta:

- **1xx:** Aceptación provisional, petición recibida y procesándose.
- **2xx:** Aceptación completa.
- **3xx:** Redirección, se necesitan realizar otras acciones para completar la petición.
- **4xx:** Error del cliente, en la sintaxis de la petición o esta no se puede completar.
- **5xx:** Error del servidor, el servidor no puede completar una petición válida.
- **6xx:** Fallo global, la petición no se puede satisfacer en ningún servidor.

2.4.2. Elementos de SIP y funcionamiento

El protocolo SIP establece una serie de elementos dentro de su red de comunicación para el correcto funcionamiento del protocolo, estos elementos son:

- **Agente de usuario (UA):** es una entidad lógica que actúa como cliente y servidor para aceptar y emitir mensajes SIP, se sitúa en los extremos del protocolo y es la entidad que interactúa con el usuario o aplicación final.
- ***Registrar* o servidores de registro:** es un servidor dentro de un dominio que acepta peticiones del tipo REGISTER, almacena la información de la dirección IP y las URL del agente de usuario que ha realizado la petición de forma que se pueda determinar la dirección física de esas URL a través del servicio de localización.
- **Servidores proxy:** Es una entidad intermedia que actúa como servidor y cliente para encaminar los mensajes SIP que recibe hacia su destino o para informar al agente de usuario hacia donde debe redirigir su mensaje.

En la Figura 3 se muestra un ejemplo del funcionamiento general del protocolo, en el ejemplo se observan los siguientes elementos:

- **Agentes de usuario (UA):** el primero de ellos pertenece al usuario “carol” el cual se encuentra en la máquina con identificador cube2214a. El segundo pertenece al usuario “bob”.
- **Servidor proxy:** retransmite los mensajes del agente de usuario de “bob”.
- **Registrar:** contiene un servicio que realiza un mapeo entre los identificadores de usuario y su localización.

El servidor proxy y el registrar son entidades lógicas y podrían estar situadas en la misma máquina pero se considera que el diagrama es más claro de este modo.

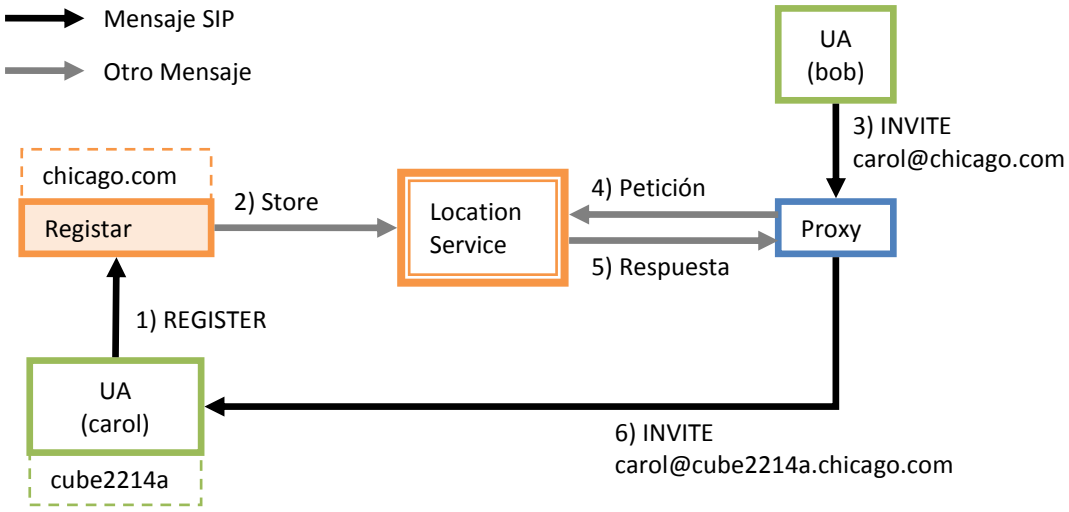


Figura 3: Ejemplo de funcionamiento del protocolo SIP

A continuación se describe la secuencia de acciones que se realizan en el diagrama:

1. El usuario “carol” inicia su agente de usuario en la máquina cube2214a e inicia el registro hacia el *register* que se encuentra en el servidor de chicago.com.
2. El servidor de registrar almacena que el usuario “carol” cuyo identificador SIP es carol@chicago.com se encuentra en la máquina cube2214a.
3. El usuario “bob” quiere iniciar una conversación con “carol” e intenta establecer una sesión con carol@chicago.com.
4. El proxy recibe la petición de “bob” y consulta el servicio de localización en qué dirección se encuentra carol@chicago.com.
5. El servicio de localización encuentra al usuario carol@chicago.com e informa que se encuentra en la dirección cube2214a.
6. El proxy retransmite la invitación hacia la máquina del usuario “carol” indicando en la dirección SIP su localización completa.

2.5. Introducción al protocolo SIMPLE

2.5.1. ¿Qué es SIMPLE?

SIMPLE es un protocolo a nivel de aplicación definido por IETF en varios RFC, se diseñó para otorgar funcionalidades de mensajería instantánea y de manejo de información de presencia al protocolo SIP. A continuación se muestran los RFCs más importantes en que se divide el protocolo y las funcionalidades que incluyen.

2.5.2. RFC 3265 - Specific Event Notification

Este RFC (27) define cómo añadir al protocolo SIP la capacidad para notificar eventos a un determinado agente de usuario subscrito a las notificaciones. El funcionamiento genérico de este sistema es que un determinado agente de usuario se suscribe a algún recurso o estado de otras entidades dentro de la red y estas entidades notificarán de las acciones sobre los recursos o de los cambios de estado al agente de usuario suscrito, un ejemplo del intercambio de mensajes se muestra en la Figura 4.

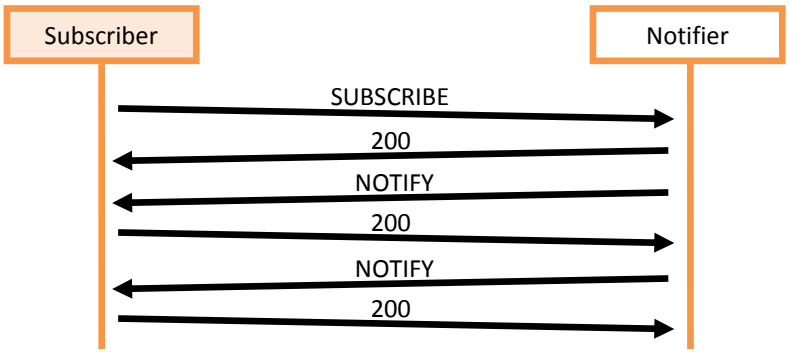


Figura 4: Ejemplo de intercambio de mensajes en una suscripción.

2.5.2.1. Nuevos métodos:

Se añaden dos tipos nuevos de métodos a los definidos por SIP:

- **SUBSCRIBE:** con este método se solicita la notificación asíncrona de un evento o grupo de eventos.
- **NOTIFY:** sirve para enviar una notificación de la ocurrencia de un evento a un elemento de SIP que se haya suscrito anteriormente a dicho evento.

2.5.2.2. Nuevas cabeceras:

Se añaden tres nuevas cabeceras a las definidas por SIP:

- **"Event":** define el evento que se está tratando, los distintos tipos de eventos no se especifican en el RFC, se indica que deben ser definidos individualmente y registrados en IANA.
- **"Allow-Eventes":** se incluye junto a una lista de los eventos que soporta el cliente.
- **"Subscription-State":** indica el estado de la suscripción.

2.5.2.3. Nuevos códigos de respuesta:

Se añaden dos nuevos códigos de respuestas a los definidos en SIP:

- **202 Accepted:** Indica que se ha aceptado la petición pero que todavía no se ha realizado la acción requerida (refiriéndose a que no se ha enviado notificación).
- **489 Bad Event:** Indica que el servidor no entiende el evento notificado en la cabecera "Event".

2.5.3. RFC 3856 - A Presence Event Package

Este RFC (28) define cómo utilizar los mensajes definidos en el RFC anterior para desarrollar un sistema y una terminología que permita construir un protocolo de presencia. Para ello propone una extensión de SIP añadiendo tres tipos de entidades que manejarán esta información y los nuevos mensajes SUSCRIBE y NOTIFY, utilizando la capacidad que tienen las redes SIP para encaminar los mensajes hacia el destinatario a partir de su dirección SIP.

2.5.3.1. Nuevas entidades:

Se añaden tres nuevas entidades a las definidas por SIP:

- **Presence User Agent (PUA):** Es un agente de usuario que manipula información presencial para una determinada dirección SIP, se permiten múltiples PUA para una misma dirección SIP pudiendo así ser utilizada la misma dirección desde distintos dispositivos.
- **Presence Agent (PA):** Es un agente de usuario que recibe peticiones de suscripción, las responden y generan los pertinentes mensajes de notificación.
- **Presence Server:** Es una entidad física que puede actuar tanto como un PA o como un proxy para las peticiones de suscripción.

2.5.4. RFC 3903 - Extension for Event State Publication

Este RFC (29) propone una extensión del framework desarrollado hasta ahora para el manejo de la información presencial, esta extensión pretende abordar la publicación de la información presencial a un agente que se encargue de componer, mantener y distribuir esta información.

2.5.4.1. Nuevo método:

- **PUBLISH:** se utiliza para publicar el estado a una entidad responsable de crear las notificaciones.

2.5.4.2. Nuevas cabeceras:

Se añaden dos nuevas cabeceras a las definidas por SIP:

- **"SIP-ETag":** permite identificar unívocamente un mensaje PUBLISH aceptado.
- **"SIP-If-Match":** indica que la petición que se está realizando hace referencia al mensaje PUBLISH que tiene el mismo `eTag`.

2.5.4.3. Nuevo código de respuesta:

- **412 Conditional Request Failed:** Indica que la precondición dada para la petición ha fallado.

2.5.5. RFC 3863 - Presence Information Data Format

Este RFC (30) define un formato común para los datos de información presencial (PIDF) que sean compatibles con protocolos CPP permitiendo de este modo transmitir esta información entre distintos protocolos sin que se realice ninguna modificación, mejorando así la seguridad y el rendimiento.

Para hacer referencia a este tipo de dato se ha asignado un nuevo tipo de medio para la cabecera `Content-Type` siendo el nuevo valor `application/pidf+xml`.

Este modelo se codifica en XML bien formado por lo que debe incluir una declaración de codificación. En la Tabla 11 se muestra un ejemplo y posteriormente se describen sus partes.

Tabla 11: Ejemplo de documento PIDF

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
                                        entity="pres:someone@example.com">
  <tuple id="sg89ae">
    <status>
      <basic>open</basic>
    </status>
    <contact priority="0.8">tel:+09012345678</contact>
  </tuple>
</presence>
```

2.5.5.1. El elemento <presence>

El elemento <presence> puede tener cualquier número de elementos <tuple> seguido de cualquier número de elementos <note> y seguido de cualquier número de cualquier número de extensiones opcionales.

El elemento <presence> debe tener un atributo 'entity' que defina la URL del *PRESENTITY* que publica este documento de presencia, también debe contener un atributo 'xmlns' para indicar el *namespace* en que se utiliza el documento, puede contener otros atributos para indicar otras declaraciones de *namespace* usadas en extensiones.

2.5.5.2. El elemento <tuple>

El elemento <tuple> debe tener un elemento obligatorio <status> seguido de cualquier número de extensiones opcionales, seguido de un elemento opcional <contact>, seguido de cualquier número de elementos <note> y seguido de un elemento opcional <timestamp>.

El elemento <tuple> debe tener un atributo 'id' que se utiliza para distinguir las distintas tuplas para un mismo *PRESENTITY* siendo el valor un identificador univoco de tipo *String*.

2.5.5.3. El elemento <status>

El elemento <status> puede tener un elemento opcional <basic> seguido de cualquier número de extensiones opcionales teniendo como única restricción que al menos debe de haber un elemento hijo.

2.5.5.4. El elemento <basic>

El elemento <basic> debe contener uno de los valores siguientes "open" o "closed":

- **open:** significa que el cliente asociado está listo para recibir menajes.
- **closed:** significa que el cliente no está listo para recibir mensajes.

2.5.5.5. El elemento <contact>

El elemento <contact> contiene la dirección URL del contacto.

El elemento <contact> puede tener un atributo opcional 'priority' que indica la prioridad de esta dirección, el valor de este atributo debe ser un valor en el intervalo [0-1] con un máximo de 3 decimales.

2.5.5.6. El elemento <note>

El elemento <note> contiene un *String* que se utiliza para comentarios.

El elemento <note> debería tener un atributo opcional 'xml:lang' para especificar el idioma del comentario.

2.5.5.7. El elemento <timestamp>

El elemento <timestamp> contiene un *String* que indica la fecha y la hora en el que se produjo el cambio de estado de esta tupla, este valor debe seguir el formato IMPP.

2.5.6. RFC 3428 - Extension for Instant Messaging

Este RFC (31) propone cómo aportar la funcionalidad de mensajería instantánea al protocolo SIP, entendiendo mensajería instantánea como el intercambio de texto generalmente de poca longitud que no necesita ser almacenado.

2.5.6.1. Nuevo método:

- **MESSAGE:** se utiliza para mandar mensajes de texto, debe contener un cuerpo de mensaje y las cabeceras estándar MIME para identificar el contenido.

2.6. Introducción a J2ME

La tecnología J2ME (32) nace a partir del JSR68 definido por Java Community Process (JCP); se creó con el objetivo de ser capaz de funcionar en pequeños dispositivos con restricciones en el tamaño de memoria y la capacidad de potencia. La plataforma consiste en una colección de tecnologías y especificaciones que se combinan para crear una máquina virtual completa capaz de cubrir las necesidades del dispositivo en el que se incluye. Esta tecnología está basada en 3 elementos:

- **Una configuración:** que incluye el conjunto básico de librerías y las capacidades iniciales de la máquina virtual para un amplio rango de dispositivos.
- **Un perfil:** que incluye el conjunto de API's que soportan un rango limitado de dispositivos.
- **Un paquete opcional:** que es un conjunto de API's que definen tecnologías específicas.

Durante el desarrollo de la plataforma Java ME han surgido dos configuraciones, la primera *Connected Limited Device Configuration* (CLDC) más limitada cubre el rango de los dispositivos pequeños, la segunda *Connected Device Configuration* (CDC) cubre el rango de dispositivos con mayor potencia como pueden ser *smart-phones* y set top boxes. La Figura 5 ofrece una visión general de las distintas tecnologías de Java.

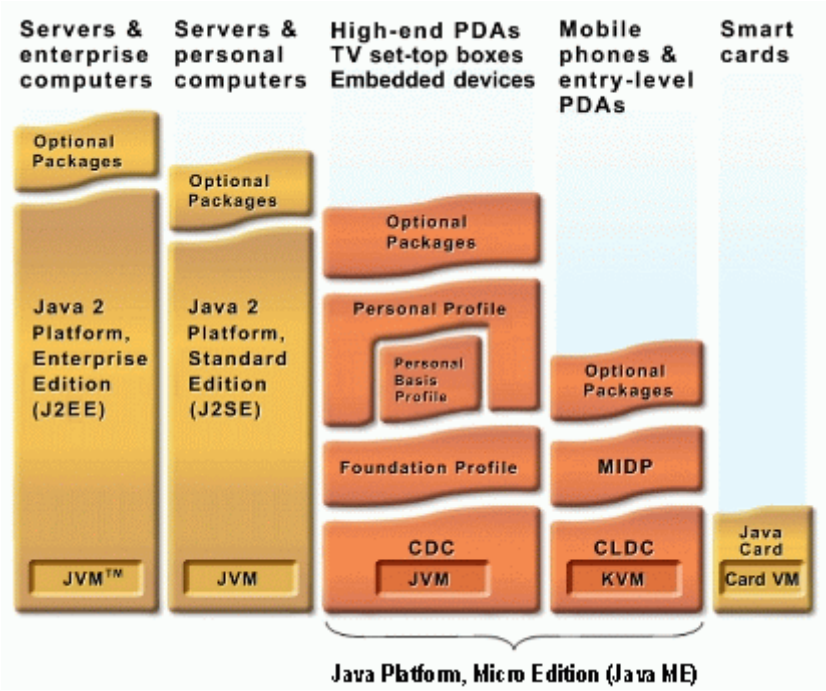


Figura 5: *The Java Platform*(32)

Esta tecnología ha tenido una gran aceptación en el entorno de los móviles y los fabricantes han incluido una implementación del mismo en sus terminales. Debido a que la plataforma se define como un conjunto de librerías es necesario una especificación del conjunto de las mismas que deben incluir los terminales; para ello se define el *Mobile Service Architecture* (MSA) que especifica las librerías obligatorias que debe implementar todo terminal estas librerías se pueden ver en la Figura 6.

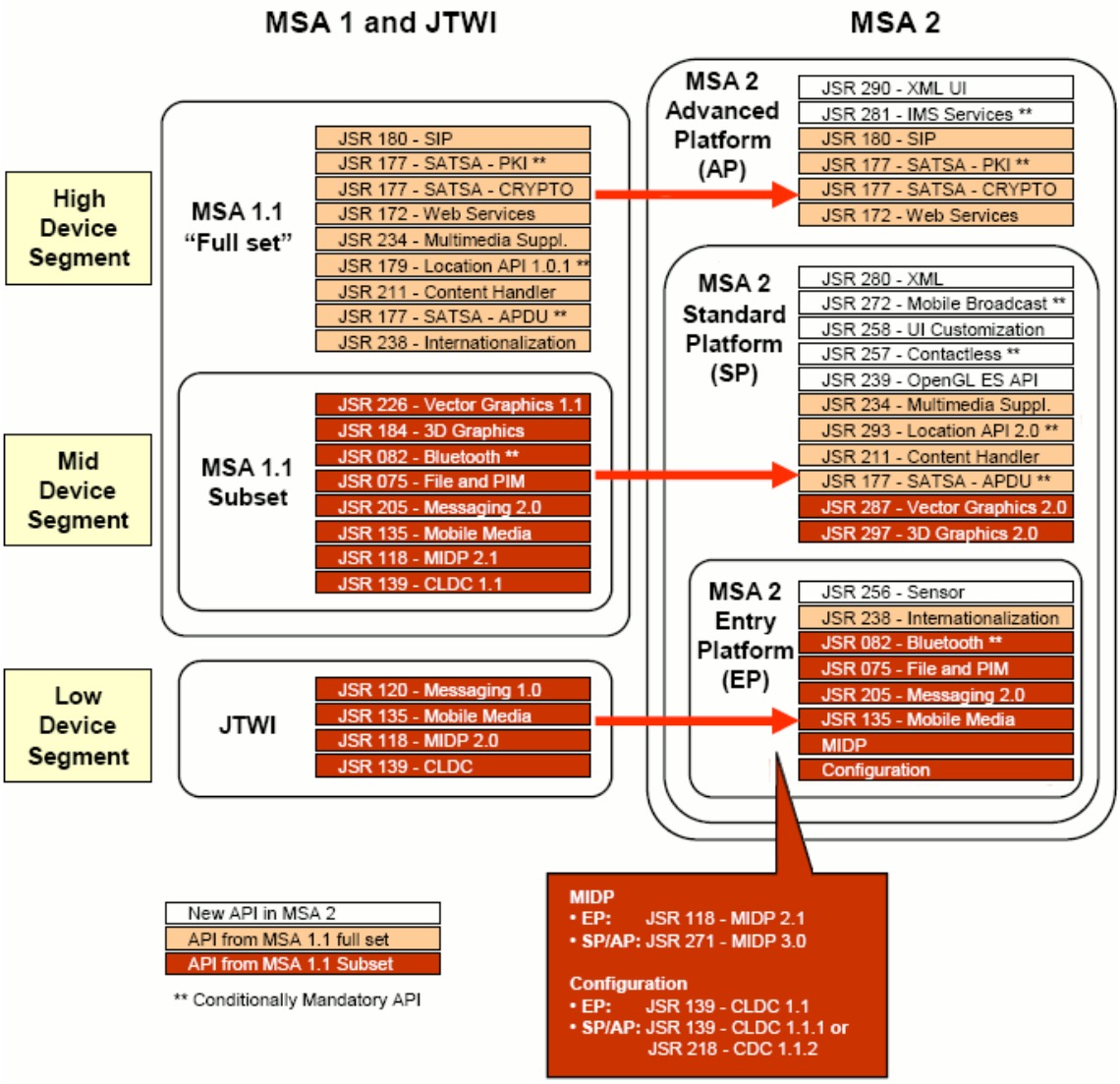


Figura 6: Especificación MSA de librerías (33)

Las conexiones SIP se soportan en el entorno J2ME a través del JSR 180, el cual se debe incluir en los terminales que implementan el *Full Set* del MSA versión 1 y dentro del MSA versión 2 se incluye dentro de las *Advanced Platform*.

2.6.1. ¿Qué es el JCP?

El JCP o *Java Community Process* (34) apareció en 1998, consiste en el mecanismo que se utiliza para la definición y el desarrollo de especificaciones técnicas para la tecnología Java. Este proceso está soportado por una comunidad abierta a la que se puede unir una parte interesada sin restricción. Estos miembros participan en el proceso de revisión aportando retroalimentación para los JSR en los que participan, también pueden formar parte del grupo de expertos que definen un JSR o incluso proponer nuevos JSR.

2.6.2. ¿Qué es un JSR?

Un JSR o *Java Specification Request* son los documentos formales que desarrolla el JCP, estos documentos proponen el desarrollo de una nueva especificación o la revisión de una especificación existente. Se presentan por uno o varios miembros a la *Program Management Office* (PMO) el cual se encarga de supervisarlos y aceptarlos. Tras este proceso los miembros que iniciaron el nuevo JSR deben formar un grupo lo suficientemente extenso y diversificado de expertos que formen el *Expert Group* destinado a trabajar en la especificación.

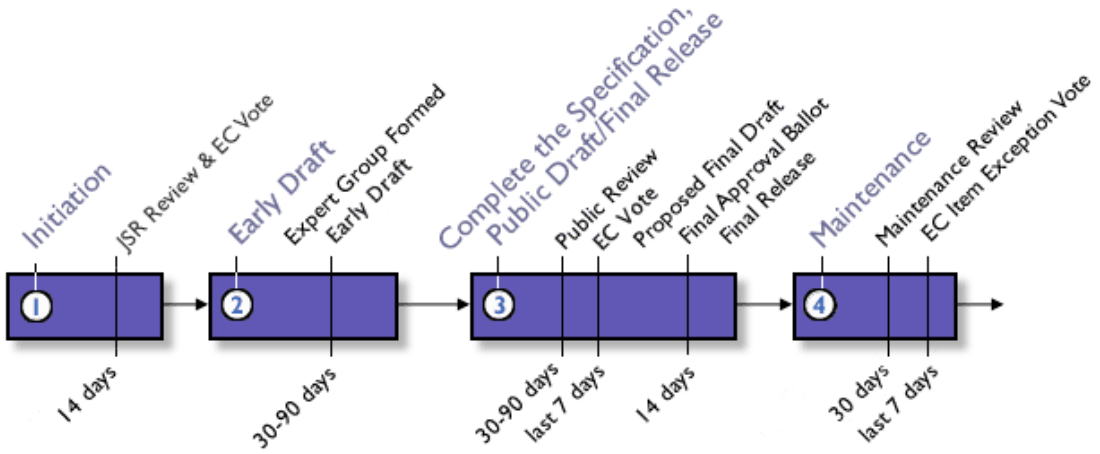


Figura 7: Ciclo de vida de un JSR(35)

Los JSR incluyen amplias especificaciones tanto de las propias tecnologías de Java (J2EE, J2ME, etc.) como a librerías específicas para una tecnología concreta como pueden ser los JSR180 para conexiones SIP en J2ME o JSR164-165 que especifican el protocolo SIMPLE siendo especificaciones genéricas y no formar parte del entorno J2ME.

Capítulo 3.

Desarrollo de la librería SIMPLE

3.1. Análisis

Se especifica el análisis de las necesidades de la librería, determinando las capacidades que debe tener y los estándares utilizados.

3.1.1. Determinación del alcance del sistema

Se desea crear una librería para J2ME que permita incorporar la funcionalidad del protocolo SIMPLE. Este protocolo sirve para realizar mensajería instantánea y para manejar la persistencia de los usuarios, su desarrollo se realiza sobre el protocolo SIP, los RFCs implicados en este protocolo son los siguientes:

Presencia:

- Núcleo del mecanismo: RFC 3265, RFC 3856, RFC 3903.
- Codificación de la presencia: RFC 3863.

Mensajería instantánea:

- Page Mode: RFC 3428.

La implementación de la librería se realizará siguiendo los estándares definidos por la comunidad Java para la realización de conexiones SIMPLE, estos estándares no son específicos de la plataforma J2ME:

- JSR 164: SIMPLE Presence (36)
- JSR 165: SIMPLE Instant Messaging (37)

3.1.1.1. Funcionalidades fuera del alcance del sistema

La librería no incluirá en su funcionalidad las partes opcionales de los RFCs implementados así como tampoco incluirá la definición del protocolo SIP ya que los propios terminales deben incluir la capacidad de realizar este tipo de conexiones, en el caso que nos ocupa esta librería se ofrece a partir del JSR 180 (38).

3.1.1.2. Ventajas de esta librería

La principal ventaja de la utilización de esta librería es que forma parte de un estándar libre mantenido por la comunidad IETF lo que ofrece las siguientes características:

Soporte y compatibilidad a largo plazo

Evita la mala gestión del ciclo de vida del software destinado a dejar obsoleto el software. Estos estándares se diseñan para tener un amplio ciclo de vida.

Formatos estándar

Al regirse por formatos estándar de comunicación y transmisión de datos se facilita la interoperabilidad de los sistemas, lo que evita incompatibilidades si se implementan correctamente.

Sistemas sin puertas traseras y más seguros

El acceso a la especificación del estándar permite realizar auditorías del mismo evitando puertas traseras y detectando con mayor facilidad problemas de seguridad.

Otra ventaja de la librería es que se implementa en un lenguaje de alto nivel multiplataforma, lo que permite exportar con relativa facilidad el código a otros sistemas operativos de dispositivos portátiles que dispongan de una máquina virtual Java con J2ME.

Por último cabe destacar que la utilización de los estándares facilita la creación de agentes de usuarios específicos sin la necesidad de incorporar toda la funcionalidad.

3.1.2. Especificación de estándares y normas

Para la especificación de los diagramas tanto del análisis como del diseño del sistema de la información, se utilizará el Lenguaje Unificado de Modelado o *Unified Modeling Language* (UML)(39).
Se deben cumplir las especificaciones de los estándares que se están utilizando e implementando, por ello se debe respetar la terminología definida en los RFC que se utilizan e interpretándola como indica el RFC 2119 (40). Las pautas opcionales referenciadas no se implementan.

3.1.2.1. Restricciones generales

La librería podrá ser utilizada en terminales que dispongan de soporte para la máquina virtual de Java en su versión de J2ME CLDCv1.1 y de una implementación de la librería JSR180.

3.1.2.2. Entorno operacional

La librería realizada se ejecutará como se ha indicado en terminales que dispongan de una máquina virtual Java J2ME y que implementen el JSR180. Actualmente se dispone de simuladores y de una serie de terminales(41) (42) con estas características:

Simuladores

Cualquier IDE con soporte para J2ME y el *wireless toolkit* de Java u otra plataforma compatible.

Terminales LG

KC910E	GR500	KC910	LG385	LX260/LG260
KC91I	KS10	KC910QA	LX600	LX570/LG570
KT610	GD880	KC910Q		

Terminales Sony Ericsson

JP-8	JP-8.3	JP-8.4	JP-8.5	C902

Terminales Blackberry

8310

Terminales Samsung

SGH-I560

Terminales Nokia

X5-01	E73 Mode	E5-00	C5-00	6700 slide
6760 slide	6790 slide	6790 Surge	E72	6730 classic
E52	E71x	N86 8MP	E55	5730 XpressMusic
6720 classic	E75	E63	N96-3	6710 Navigator
N85	N79	E66	E715	5630 XpressMusic
6650 fold	6124 classic	N78	N96	320 XpressMusic
6220 classic	N82E51	N95-3 NAM	N81	6210 Navigator
N81 8GB	N95 8GB	6121 classic	6120 classic	5700 XpressMusic
N77	E61i	E65	N76	6110 Navigator
N93i	6290	N91 8GB	N95	E90 Communicator
N75	E62	E50	5500 Sport	N93
N73	N92	N71	N80	E70
E61	E60	3250	N91	

3.1.3. Establecimiento de los requisitos

La librería debe implementar las API's definidas en los JSR 164 y JSR 165 por lo que no se dispone de requisitos.

3.2. Diseño

Se especifica el diseño de la librería, incluyendo arquitectura, clases y el diseño detallado.

3.2.1. Tecnologías utilizadas

Para el desarrollo de la librería se utiliza Java en su versión J2ME, esta tecnología está orientada a la ejecución en dispositivos móviles incluyendo gran parte de la funcionalidad ofrecida por otras versiones de Java y adaptándolas a las necesidades de estos dispositivos.

También se utiliza la librería JSR180 que define cómo realizar conexiones SIP, esta librería sigue la estructura de clases definida en la Figura 8. Como se puede apreciar esta librería permite definir conexiones clientes y servidor, así como ofrece una serie de clases de ayuda para implementar toda la funcionalidad de un agente de usuario.

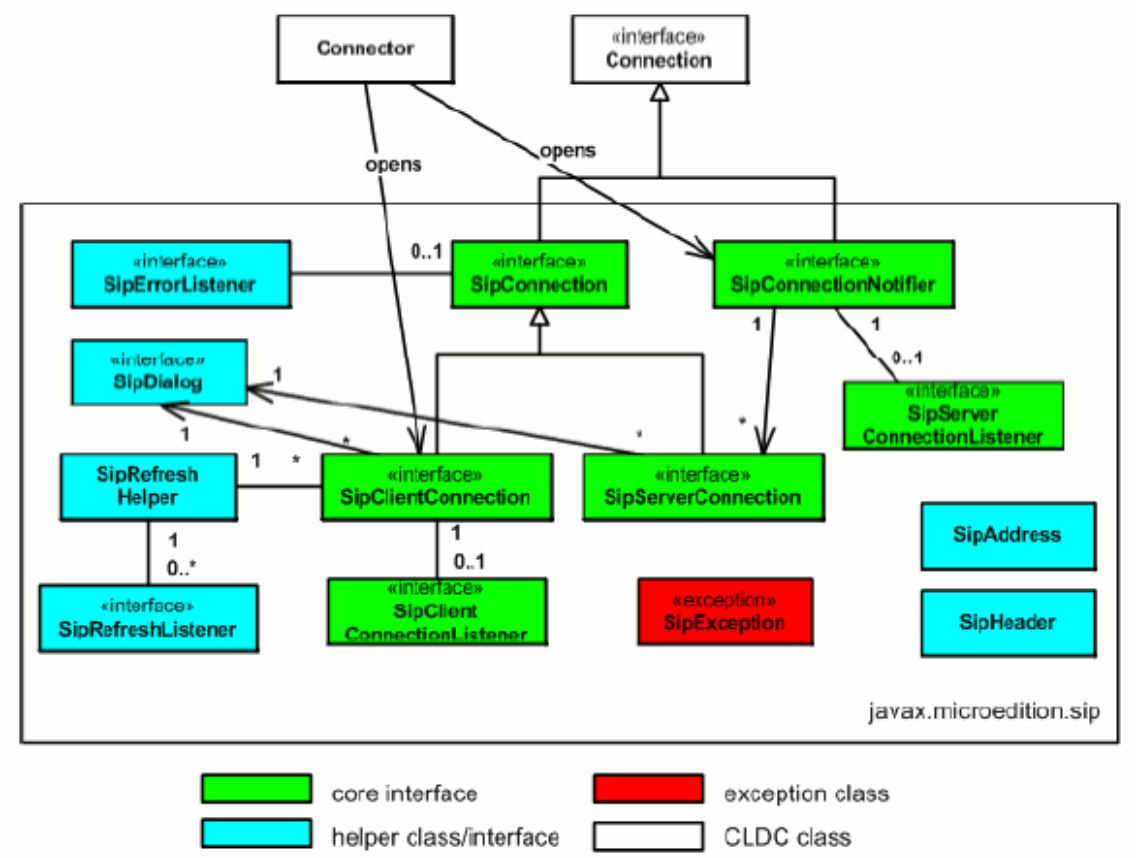


Figura 8: Diagrama de clases de la librería JSR 180 (38)

3.2.2. Arquitectura

La librería se especifica como parte de la arquitectura por capas de los protocolos de comunicación y se sitúa directamente encima de la librería JSR180 tal y como se muestra en la Figura 9. De este modo las aplicaciones que desean realizar una implementación del protocolo SIMPLE se abstraen de la implementación de conexiones SIP.

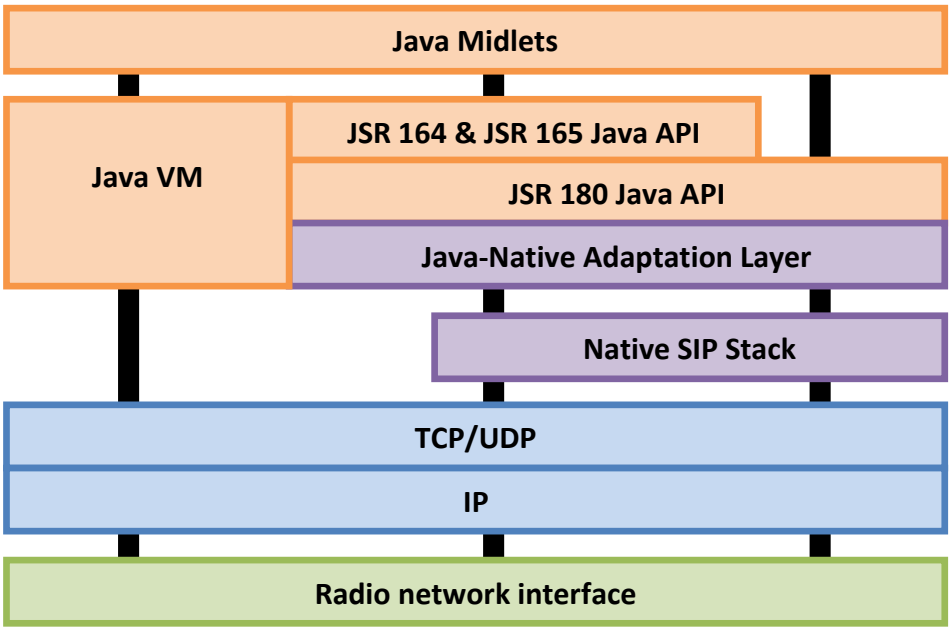


Figura 9: Arquitectura de la librería SIMPLE

La arquitectura define la capa en el dominio de comunicación en el que se define la lógica de creación de distintos elementos del protocolo SIMPLE como las direcciones, las cabeceras, los distintos agentes de usuario, los mensajes y los documentos de especificación de estado.

La implementación de la capa se encarga de especificar las cabeceras básicas de cada uno de los distintos tipos de peticiones y respuestas enviadas.

Por último se realiza un mantenimiento de las transacciones que realizan los agentes de usuario así como el estado de las mismas y de la lógica de actuación ante cada una de las peticiones y respuestas recibidas, determinando cuándo se debe realizar una notificación a la aplicación final.

3.2.3. Pautas de diseño

En el desarrollo de la librería se han seguido estrictamente las recomendaciones de la Programación Orientada a Objetos (POO), en particular los conceptos de encapsulación y polimorfismo. La librería se ha desarrollado dentro del paquete "`edu.uc3m.simple`", y se ha reducido la visibilidad de las clases y métodos al mínimo, evitando que éstas sean accesibles desde fuera del paquete en la medida de lo posible. Esta práctica es muy recomendable, ya que el uso de clases, métodos o atributos "`public`" o "`protected`" permiten su uso desde código externo, y por lo tanto estos quedarían añadidos al API público de nuestra aplicación, sólo el código "`private`" o "`package-protected`" evita la inclusión de los elementos en el API público.

Estas prácticas chocan con la costumbre extendida de utilizar los paquetes como una herramienta para organizar las clases, permitiendo hacerlo en una estructura similar a la de carpetas, pero como se ha indicado anteriormente, desde el punto de vista de la orientación a objetos, estas prácticas están completamente desaconsejadas al romper con la encapsulación proporcionada por el sistema de paquetes. Cada uno de los paquetes engloba un conjunto de elementos de código que presenta un interfaz público para la interacción con código de otros paquetes.

Por otro lado la implementación de las interfaces debería situarse en el paquete "`javax.simple`" sin embargo esto no ha podido realizarse ya que está reservado para implementaciones oficiales, por ello se han desarrollado estas interfaces en el paquete "`edu.uc3m.isimple`".

3.2.4. Diseño detallado

Se muestra el diseño de clases definido en los JSR 164 y JSR165, especificando un nivel de detalle mayor. La Figura 10 muestra una vista global del sistema incluyendo las relaciones entre las clases.

A continuación se muestra la Figura 11 con el diseño de clases que se ha realizado para la librería, en este diagrama las clases con el mismo nombre que una interfaz de la Figura 10 y con la terminación `Impl` es una realización de dicha interfaz

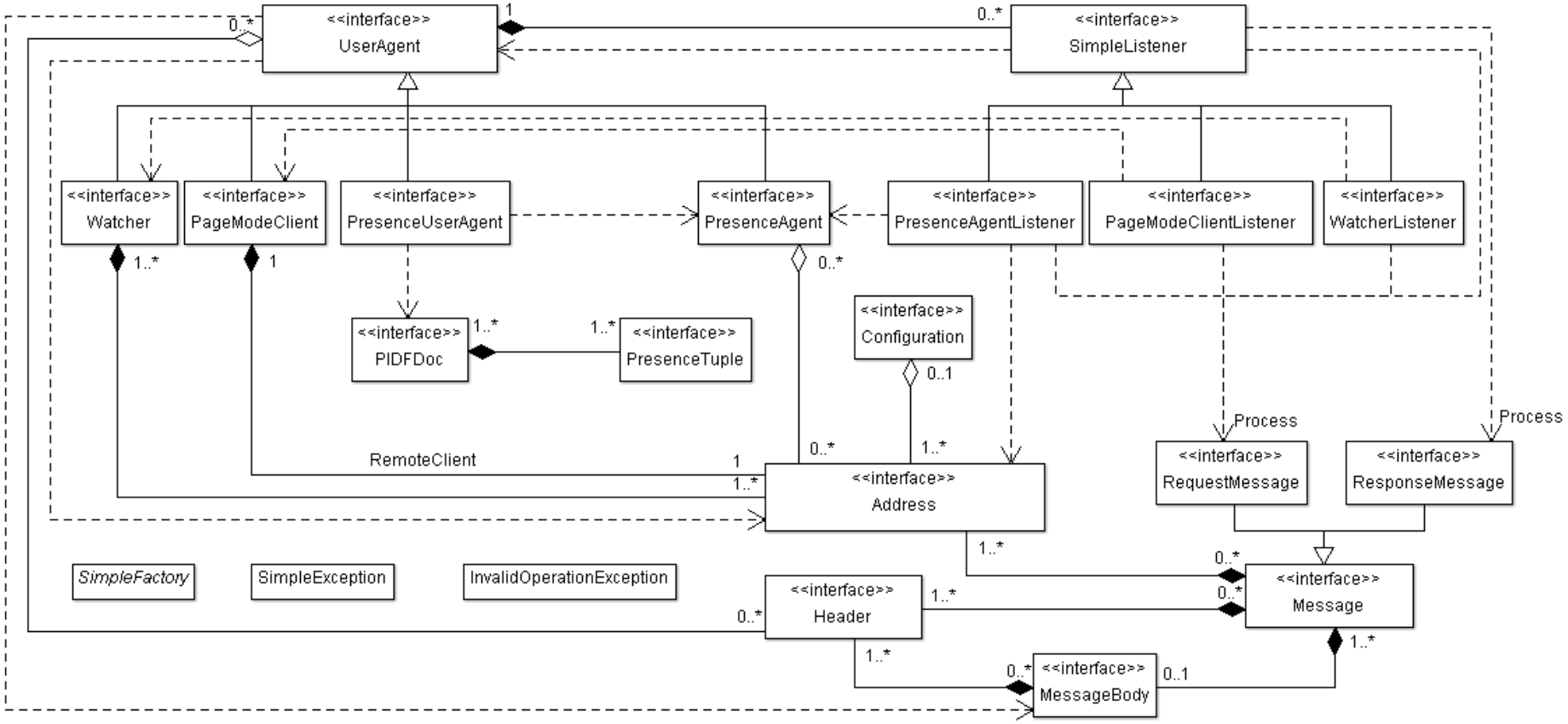


Figura 10: Diagrama de clases de los JSR 164 y JSR 165[3]

[3] En esta figura no se muestran las relaciones de `InvalidOperationException`, `SimpleException` y `SimpleFactory` ya que dificultaría la lectura del diagrama

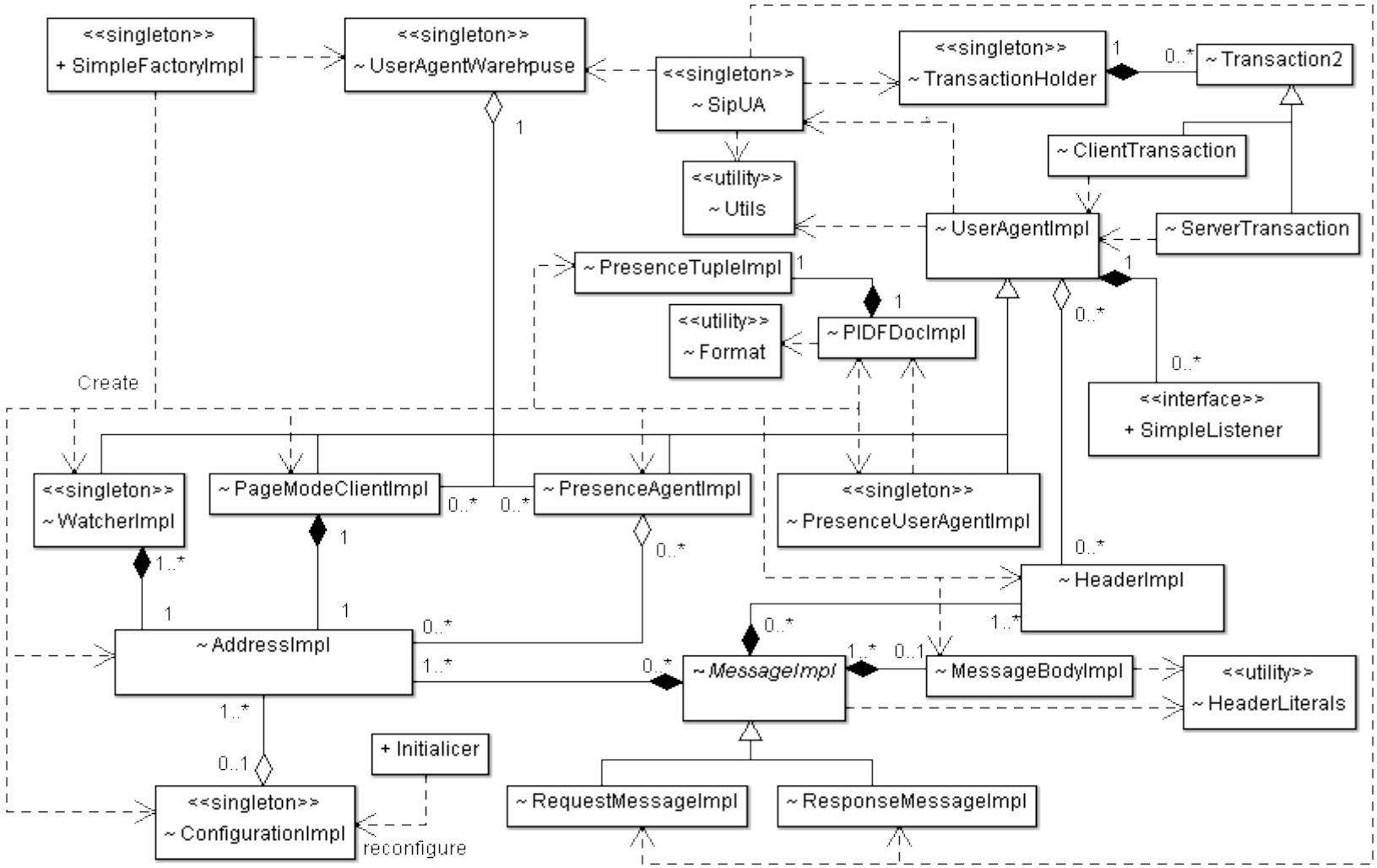


Figura 11: Diagrama de clases de la librería desarrollada

3.2.4.1. Diseño detallado de la implementación

Class AddressImpl

La clase `AddressImpl` encapsula una dirección SIP bien formada. Sobrescribe el método `equals` para realizar la comprobación de igualdad a través de la *Uniform Resource Identifier* (URI) de las direcciones. Los constructores crean y almacenan un objeto `SIPAddress`, recibiendo como parámetros una dirección SIP bien formada o un identificador de la dirección con la URI asociada

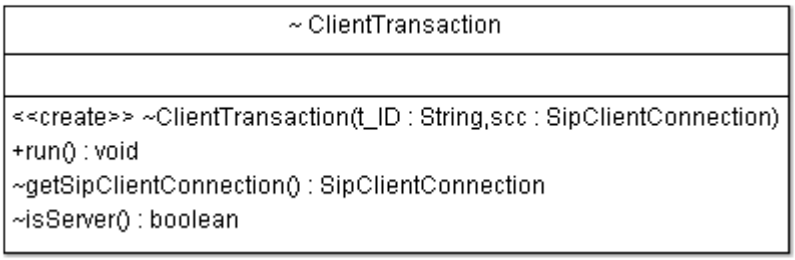
Tabla 12: Clase AddressImpl

Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.Address	**Superclase**	`Object`
Excepciones	`IllegalArgumentException`, `NullPointerException` y `ClassCastException`		

Class ClientTransaction

La clase `ClientTransaction` define una transacción de un cliente SIP del tipo non-INVITE, esta clase implementa el método abstracto *run* de `Transaction2` para añadir la funcionalidad del hilo que define; el método controla los *timer* especificados en el RFC3261 sección 17.1.2 para este tipo de transacciones apoyándose en los estados definidos en `UserAgent`.

Tabla 13: Clase ClientTransaction

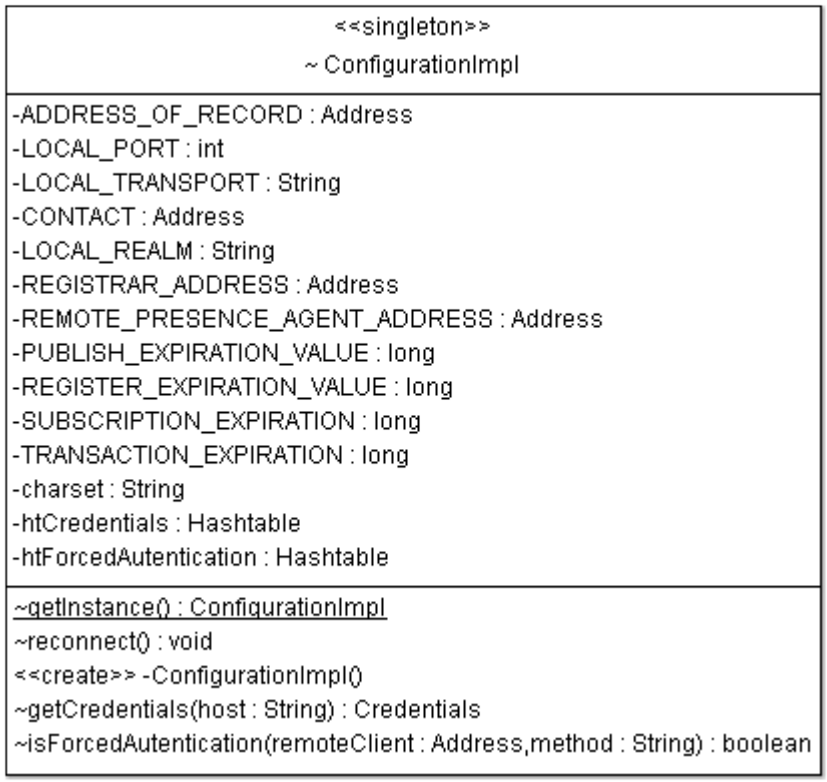


Visibilidad	Paquete	**Abstracta**	No
Interface	-	**Superclase**	`Transaction2`
Excepciones	-		

Class ConfigurationImpl

Esta clase almacena la configuración por defecto de la conexión SIP/SIMPLE y registra para su posterior obtención los credenciales de un usuario. Implementa una versión del patrón *singleton* que lanza una excepción.

Tabla 14: Clase ConfigurationImpl

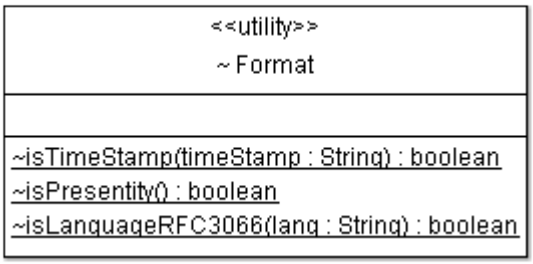


Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.Configuration	**Superclase**	`Object`
Excepciones	`InstantiationException`, `IllegalArgumentException`		

Class Format

La clase `format` debe comprobar que se cumplen los formatos correctamente.

Tabla 15: Clase Format

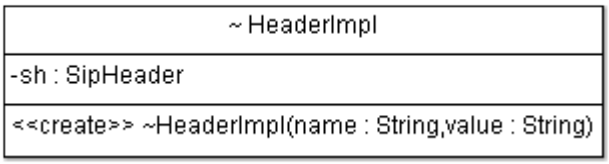


Visibilidad	Paquete	**Abstracta**	No
Interface	-	**Superclase**	`Object`
Excepciones	-		

Class HeaderImpl

La clase `HeaderImpl` encapsula una cabecera SIP bien formada, se sobrescriben los métodos `toString()` y `clone()`, el primer método se utiliza para devolver el contenido de la cabecera en el formato correcto definido por SIP para la transmisión de mensajes y el segundo método se utiliza para obtener una copia correcta de la cabecera.

Tabla 16: Clase HeaderImpl

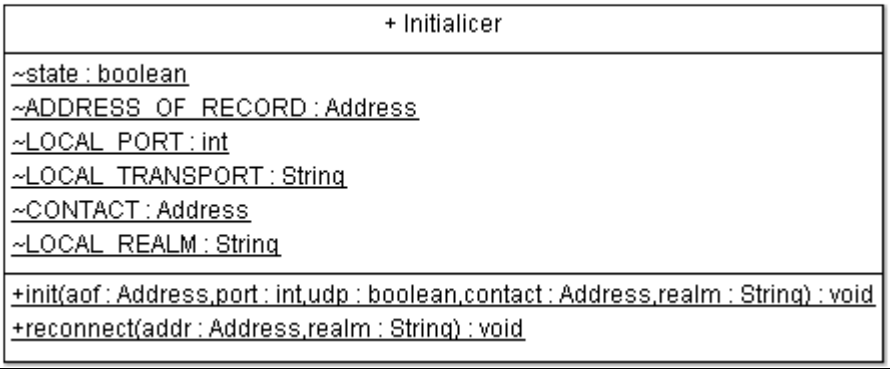


Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.Header	**Superclase**	`Object`
Excepciones	-		

Class Initialicer

La clase `Initialicer` almacena los datos de un usuario para realizar desde el exterior una reconfiguración de la clase `ConfigurationImpl`. La clase se inicia con el método `init()` en el que se le pasa la configuración básica de la conexión y del usuario, a partir de entonces se pueden cambiar los datos del usuario y los credenciales utilizando el método `reconnect()`.

Tabla 17: Clase Initialicer



Visibilidad	Pública	**Abstracta**	No
Interface	-	**Superclase**	`Object`
Excepciones	-		

Class MessageBodyImpl

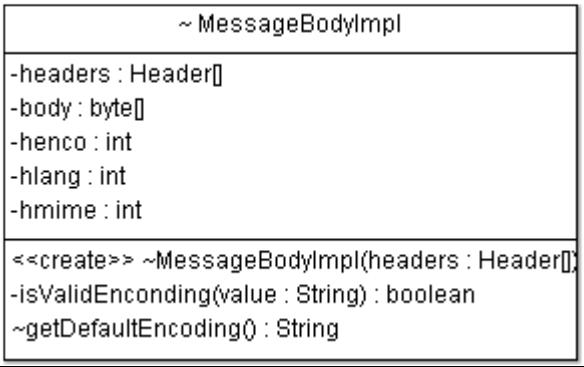
La clase `MessageBodyImple` representa el cuerpo de un mensaje SIP, el constructor de la clase recibe como parámetros las cabeceras propias definidas en MIME para la especificación del contenido del cuerpo. La propia clase comprueba que se reciben codificaciones soportadas:

- `text/`
- `application/sdp`
- `application/pidf+xml`

Al ser una realización de la interfaz `MessageBody` debería soportar la recepción y el envío de imágenes, pero esta característica no se ha soportado en esta versión de la librería por lo tanto los siguientes métodos de la clase responden con un `UnsuportedOperationException`:

- `public Object getImageBody()`
- `public void setImageBody(Object image)`

Tabla 18: Clase MessageBodyImpl

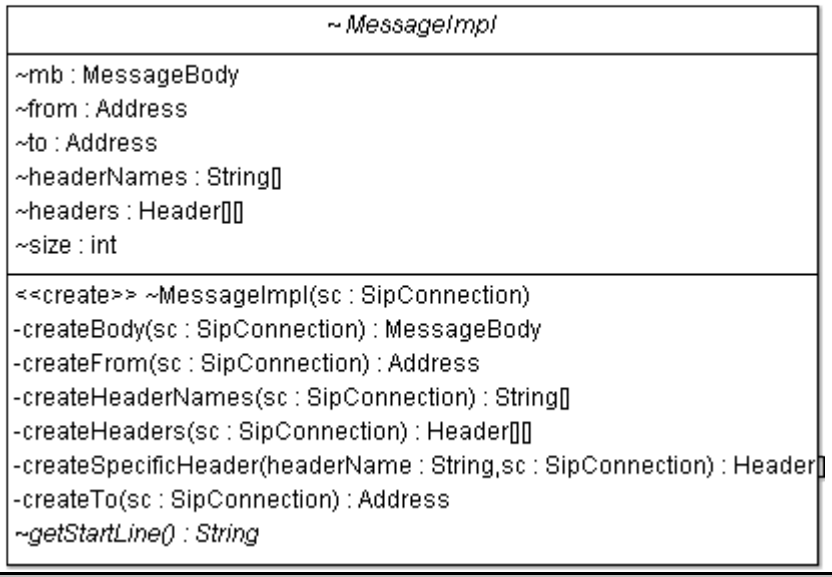


Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.MessageBody	**Superclase**	`Object`
Excepciones	`UnsupportedEncodingException, IllegalArgumentException, UnsupportedOperationException, NullPointerException`		

Class MessageImpl

La clase abstracta `MessageImpl` define la estructura de un mensaje genérico SIP, el constructor de la clase recibe una conexión SIP que almacena un mensaje y la clase realiza el mapeo de los datos para obtener el cuerpo del mismo y las cabeceras, delegando en las clases hijas el mapeo de los datos de la línea de inicio. Esta clase mantiene las restricciones de acceso a las cabeceras definidas en el JSR180.

Tabla 19: Clase MessageImpl

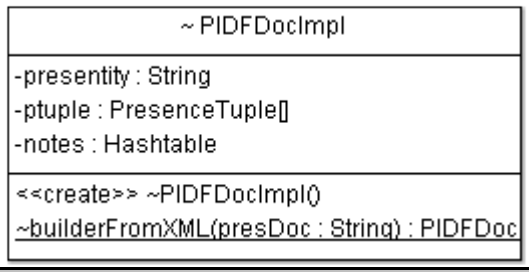


Visibilidad	Paquete	**Abstracta**	Si
Interface	edu.uc3m.isimple.Message	**Superclase**	`Object`
Excepciones	-		

Class PIDFDocImpl

La clase `PIDFDocImpl` almacena la información de presencia necesaria para crear un mensaje definido en el RFC 3863, la clase ofrece un método estático con visibilidad de paquete que se utiliza para construir objetos `PIDFDoc` a partir de una secuencia XML bien formada.

Tabla 20: Clase PIDFDocImpl

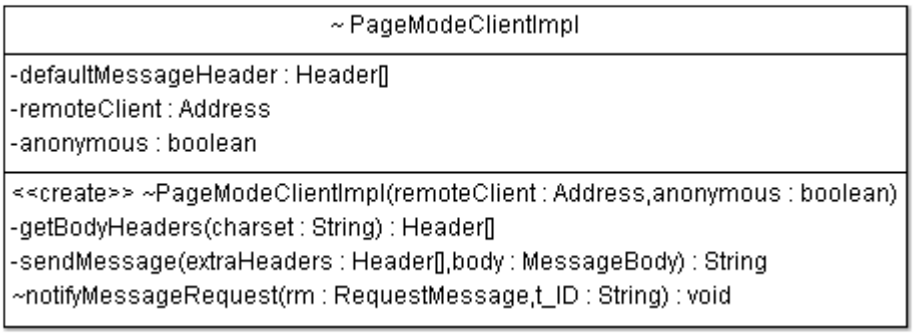


Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.PIDFDoc	**Superclase**	`Object`
Excepciones	`IllegalArgumentException`		

Class PageModeClientImpl

La clase `PageModeClientImpl` define un agente de usuario SIP encargado de enviar y recibir los mensajes entre el usuario y un contacto remoto, valiéndose para ello de la transacción creada por la clase `SipUA`. Ante la recepción de un mensaje el agente de usuario emite una respuesta 200 "Ok" e informa a la aplicación por medio del *listener* especificado para dicho agente.

Tabla 21: Clase PageModeClientImpl

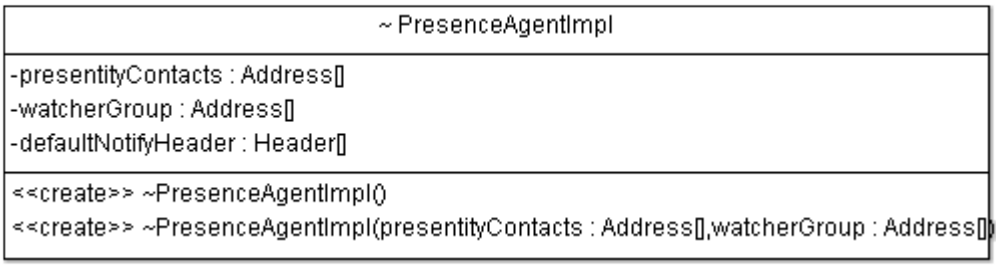


Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.PageModeClient	**Superclase**	`UserAgentImpl`
Excepciones	`SimpleException, IllegalArgumentException, UnsupportedEncodingException`		

Class PresenceAgentImpl

La clase `PresenceAgentImpl` define el agente de usuario encargado de notificar a los grupos de usuario registrados tanto externos como internos. Esta clase registra los usuarios que han pedido una petición de notificación y los `Watcher` que deben ser notificados.

Tabla 22: Clase PresenceAgentImpl

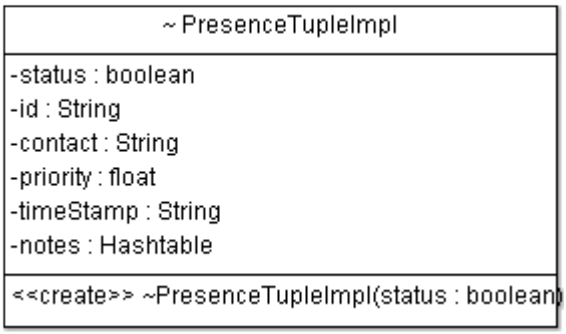


Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.PresenceAgent	**Superclase**	`UserAgentImpl`
Excepciones	`SimpleException, IllegalArgumentException`		

Class PresenceTupleImpl

La clase `PresenceTubleImpl` encapsula la información de una tupla de presencia definida en el RFC 3863.

Tabla 23: Clase PresenceTupleImpl

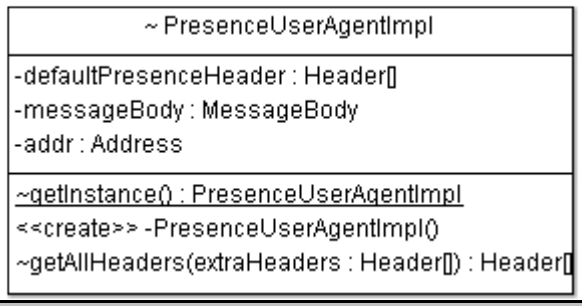


Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.PresenceTuple	**Superclase**	`Object`
Excepciones	`IllegalArgumentException`		

Class PresenceUserAgentImpl

La clase `PresenceUserAgentImpl` define un agente de usuario encargado de publicar la información de presencia de un usuario, controlando cuando se está emitiendo información nueva, cuando se está refrescando la información o cuando se está actualizando. La clase implementa el patrón *singleton* ya que debe ser única para el cliente.

Tabla 24: Clase PresenceUserAgentImpl



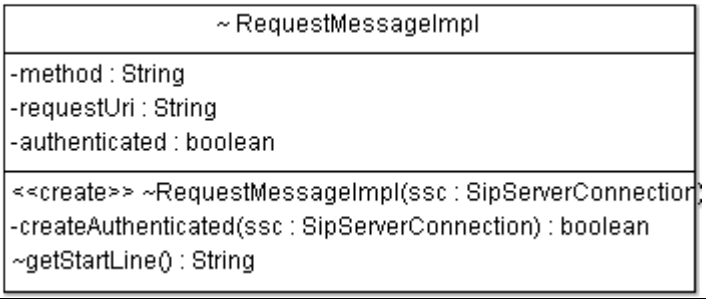
Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.PresenceUserAgent	**Superclase**	`UserAgentImpl`
Excepciones	`SimpleException, IllegalArgumentException`		

Class RequestMessageImpl

La clase `RequestMessageImpl` implementa los mensajes de petición definidos en el protocolo SIP, sobrescribe el método `getStartLine()` para obtener la primera línea como se define en el protocolo.

El constructor obtiene los datos referentes al *method* y al *requestURL* que se ha utilizado en la petición y comprueba si es un mensaje autenticado consultando la presencia de cabeceras *Authorization* o *Proxy-Authorization*.

Tabla 25: Clase RequestMessageImpl



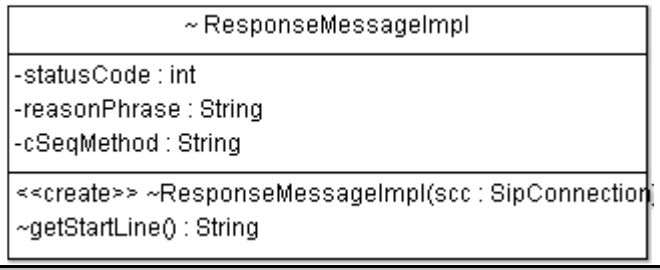
Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.RequestMessage	**Superclase**	`MessageImpl`
Excepciones	-		

Class ResponseMessageImpl

La clase `ResponseMessageImpl` implementa los mensajes de respuesta definidos en el protocolo SIP, sobrescribe el método `getStartLine()` para obtener la primera línea como se define en el protocolo.

El constructor obtiene los datos referentes al código y frase de respuesta y al método almacenado en la cabecera *CSeq* que se ha utilizado.

Tabla 26: Clase ResponseMessageImpl

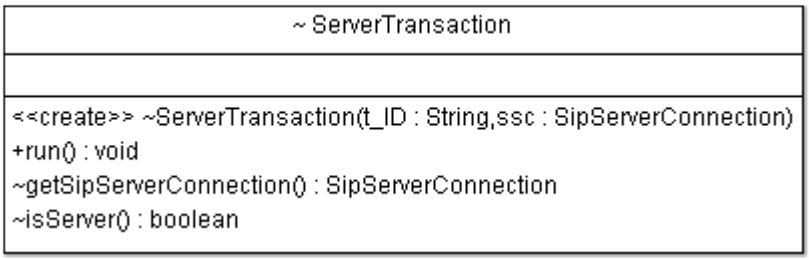


Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.ResponseMessage	**Superclase**	`MessageImpl`
Excepciones	-		

Class ServerTransaction

La clase `ClientTransaction` define una transacción de un servidor SIP del tipo non-INVITE, esta clase implementa el método abstracto *run* de `Transaction2` para añadir la funcionalidad del hilo que define; el método controla los *timer* especificados en el RFC3261 sección 17.2.2 para este tipo de transacciones apoyándose en los estados definidos en `UserAgent`.

Tabla 27: Clase ServerTransaction

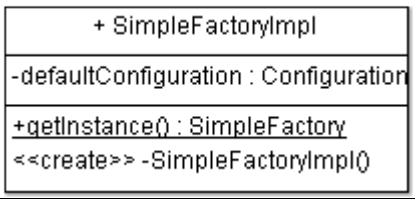


Visibilidad	Paquete	**Abstracta**	No
Interface	-	**Superclase**	`Transaction2`
Excepciones	-		

Class SimpleFactoryImpl

La clase `SimpleFactoryImpl` especifica una clase de construcción de objetos de la interfaz `edu.uc3m.isimple`, la clase es única en el cliente y especifica por ello el patrón *singleton* modificado ya que durante su creación debe obtener la instancia de `ConfiguartionImpl` y debe iniciar el *listener* de la instancia de `SipUA`

Tabla 28: Clase SimpleFactoryImpl



Visibilidad	Pública	**Abstracta**	No
Interface	-	**Superclase**	`SimpleFactory`
Excepciones	`SimpleException, IllegalArgumentException, UnsupportedEncodingException, InstantiationException`		

Class SipUA

La clase `SipUA` se encarga de la creación de las transacciones tanto de cliente como de servidor por medio de la clase `TransactionHolder` para cada uno de los distintos tipos de mensaje, para ello implementa un *listener* de mensajes SIP actuando de este modo como servidor redirigiendo las peticiones a los agentes de usuario específicos de la librería. Como cliente define las cabeceras básicas y el método que deben llevar cada uno de los mensajes e incluye la información que se le pasa desde las capas superiores. La clase se divide en operaciones de cliente o servidor SIP tal y como se indica a continuación:

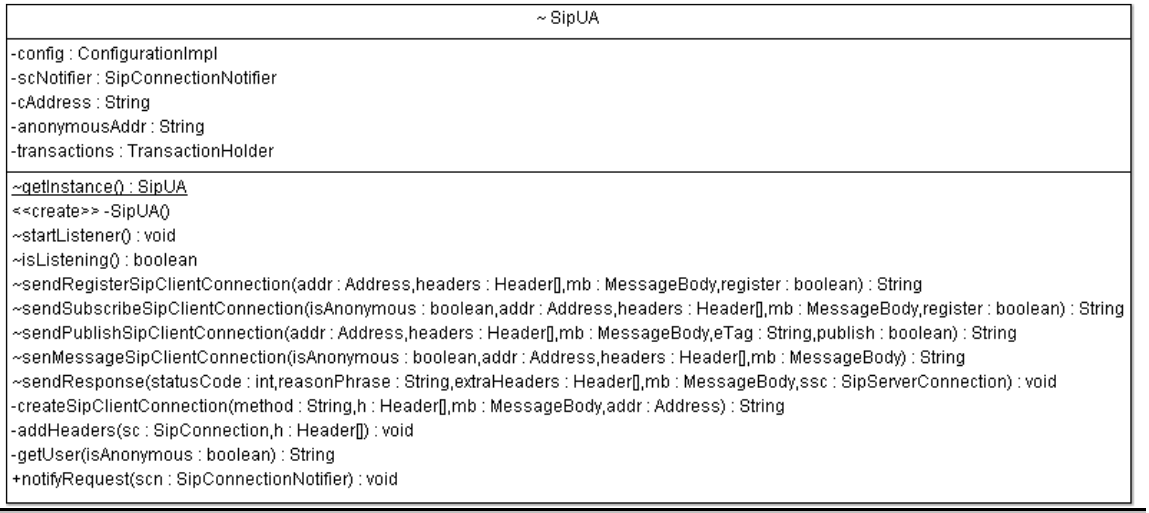
Operaciones de Cliente:

- `sendRegisterSipClientConnection()`
- `sendSubscribeSipClientConnection()`
- `sendPublishSipClientConnection()`
- `sendMessageSipClientConnection()`
- `createSipClientConnection()`
- `getUser()`

Operaciones de Servidor:

- `sendResponse()`
- `startListener()`
- `isListening()`
- `notifyRequest()`

Tabla 29: Clase SipUA



Visibilidad	Paquete	**Abstracta**	No
Interface	-	**Superclase**	`Object`
Excepciones	`IOException, SimpleException, SipException`		

Class Transaction2

La clase abstracta `Transaction2` define una transacción SIP genérica y representa un hilo de ejecución que se definirá en los hijos que la extiendan. La clase se apoya en los estados definidos en `UserAgent` que operan a nivel de bit sobre números enteros:

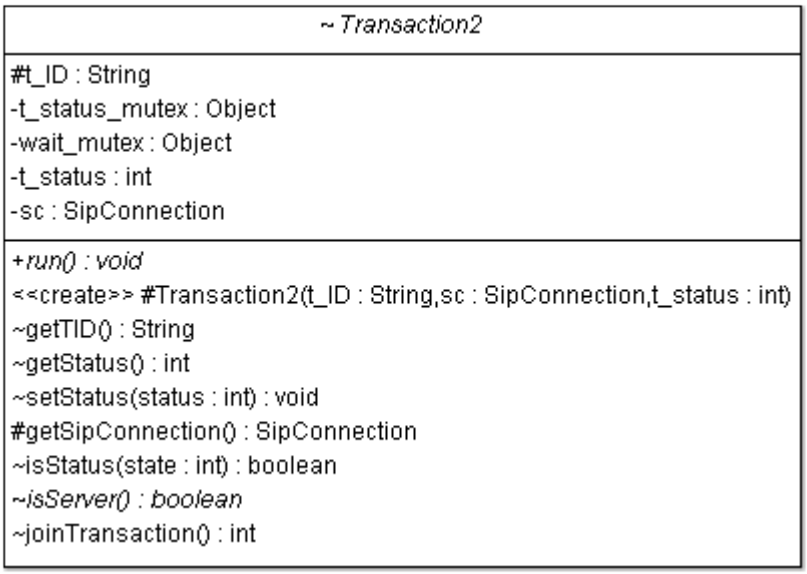
```
TRANS_IS_UNKNOWN             = 1000-1000-1000-1000-0000-0000-0000-0000
TRANS_IS_SERVER              = 0000-0000-0000-0100-0000-0000-0000-0000
TRANS_IS_FAILED              = 0010-1010-0010-1010-0000-0000-0000-0000
TRANS_IS_STATE_TRYING        = 0000-0000-0100-0000-0000-0000-0000-0000
TRANS_IS_STATE_PROCEEDING    = 0000-0001-0000-0000-0000-0000-0000-0000
TRANS_IS_STATE_COMPLETED     = 0000-0100-0000-0000-0000-0000-0000-0000
TRANS_IS_STATE_TERMINATED    = 0001-0000-0000-0000-0000-0000-0000-0000
TRANS_IS_STATE_FINISHED      = 0101-0100-0000-0000-0000-0000-0000-0000
TRANS_RESPONSE_CODE          = 0000-0000-0000-0000-1111-1111-1111-1111
```

De este modo se pueden utilizar el operador `OR` para almacenar el código de respuesta en los primeros 16 bits de un estado y recuperarlo fácilmente al realizar una operación `AND` del resultado anterior con la máscara `TRANS_RESPONSE_CODE`.

A las transacciones de servidor se le aplica al estado la máscara `TRANS_IS_SERVER` con un `OR`, después comprobando (`state & TRANS_IS_SERVER)!=0` se sabe si es servidor.

Para comprobar si se está en un estado x únicamente se realizar (`state & TRANS_IS_STATE_x)!=0`, el estado `TRANS_IS_STATE_FINISHED` es un estado especial que indica que la transacción esta completada o terminada.

Tabla 30: Clase Transaction2

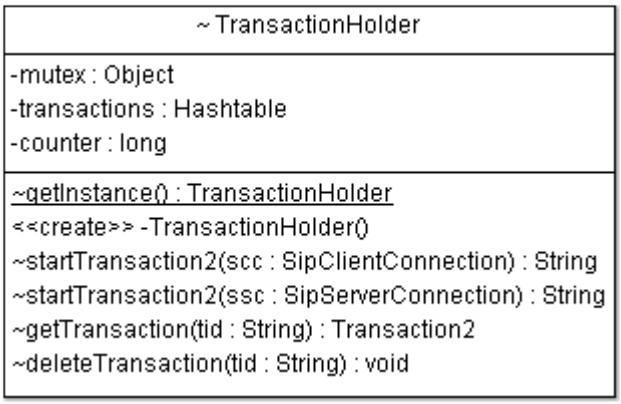


Visibilidad	Paquete	**Abstracta**	Si
Interface	-	**Superclase**	`Thread`
Excepciones	–		

Class TransactionHolder

La clase `TransactionHolder` es un manejador de `Transaction2` que las identifica y almacena por su `t_ID`, permite crear e iniciar transacciones de cliente y de servidor

Tabla 31: Clase TransactionHolder

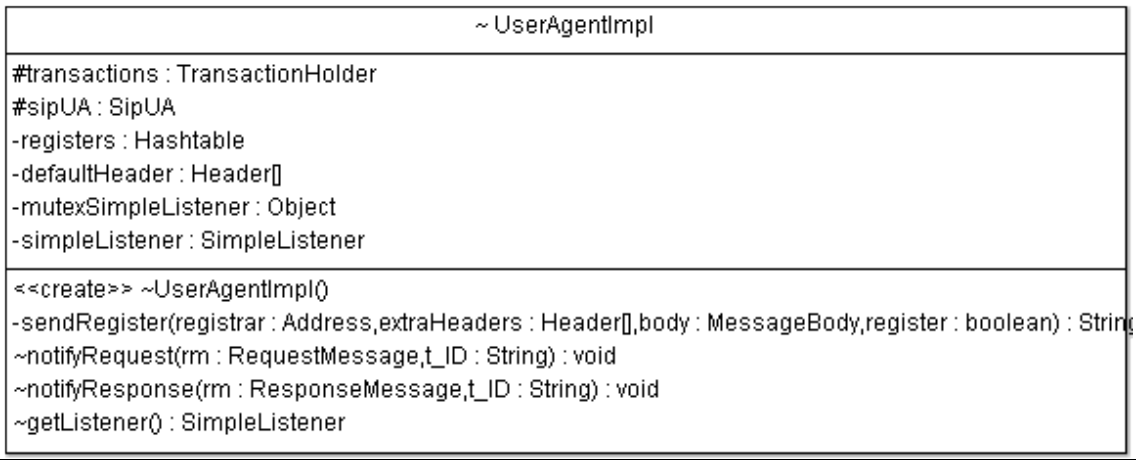


Visibilidad	Paquete	**Abstracta**	No
Interface	-	**Superclase**	`Object`
Excepciones	-		

Class UserAgentImpl

La clase `UserAgentImpl` define un agente de usuario genérico, incluye la funcionalidad básica para registrar un agente de usuario y recibir notificaciones de peticiones o respuestas, las cuales tratarán con el *listener* que se le haya asignado. La clase también gestiona las notificaciones periódicas del registro por medio de un hilo interno.

Tabla 32: Clase UserAgentImpl

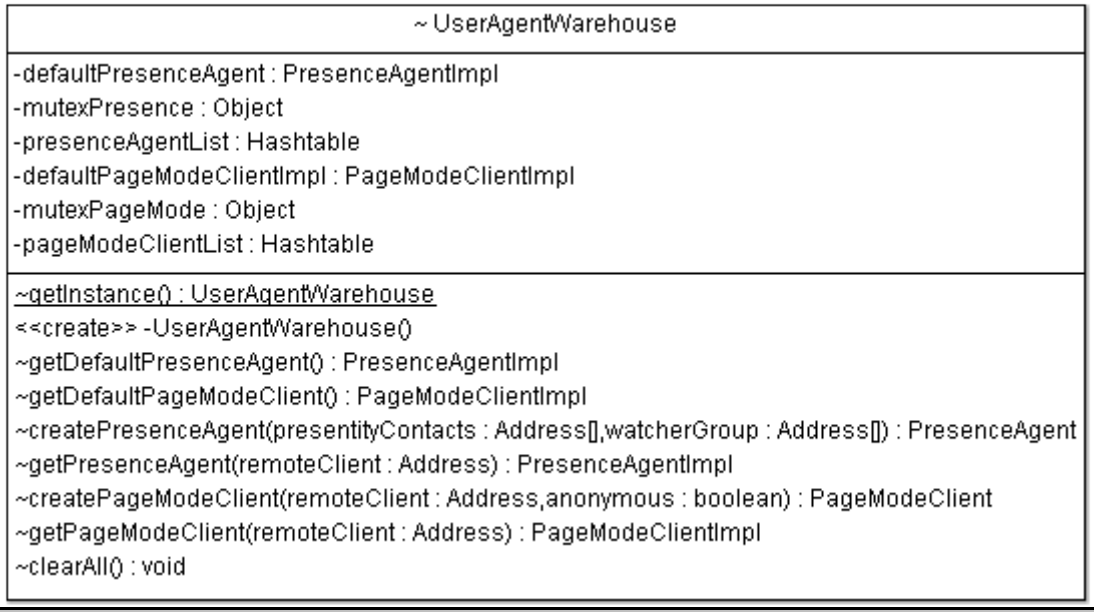


Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.UserAgent	**Superclase**	`Object`
Excepciones	`SimpleException, IllegalArgumentException`		

Class UserAgentWarehouse

La clase `UserAgentWarehouse` define un almacén de agentes de usuario, que permite recuperar los agentes de usuario por defecto y crear u obtener los específicos que gestionan la comunicación con un contacto determinado.

Tabla 33: Clase UserAgentWarehouse

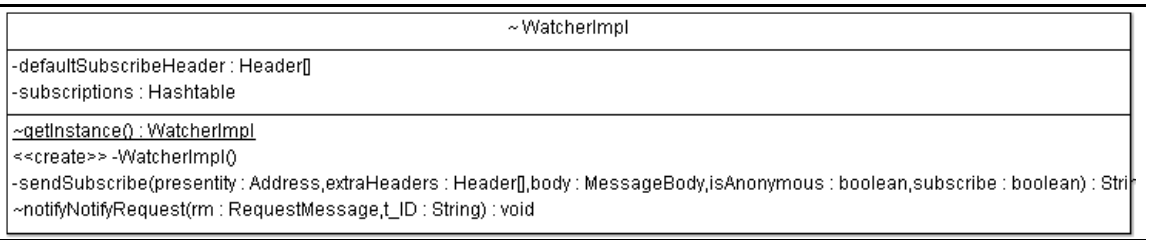


Visibilidad	Paquete	**Abstracta**	No
Interface	-	**Superclase**	`Object`
Excepciones	-		

Class WatcherImpl

La clase `WatcherImpl` define el agente de usuario encargado de realizar las subscripciones a otros usuarios y de recibir las notificaciones de información presencial, gestiona con un hilo interno el reenvío automático de las peticiones de subscripción. Al ser una clase de instanciación única implementa el patrón *singleton*.

Tabla 34: Clase UserAgentWarehouse



Visibilidad	Paquete	**Abstracta**	No
Interface	edu.uc3m.isimple.Watcher	**Superclase**	`UserAgent`
Excepciones	-		

3.2.5. Diagramas de secuencia

A continuación se recoge un conjunto de diagramas de secuencia que ilustran la interacción de los diferentes componentes de la aplicación. Cada uno de estos diagramas muestra la actividad para alguna acción de una aplicación final.

3.2.5.1. Obtener la factoría de SIMPLE

La Figura 12 detalla los pasos que se realizan cuando se obtiene por primera vez la instancia única de `SimpleFactoryImpl`. A continuación se detallan los pasos seguidos:

- La primera invocación del método `SimpleFactoryImpl.getInstance()` obtiene la instancia única de la clase `ConfigurationImpl`, al ser la primera llamada se utiliza el método `reconnect()` para obtener los parámetros de configuración.
- Se obtiene la instancia única de `SIPUA` con sus atributos y se comprueba si ya está recibiendo conexiones entrantes con el método `isListening()`.
- Si no está recibiendo conexiones se crea un `SIPConnectionNotifier` y se invoca el método `Connector.open("sip:*;type=\"application/schat\"")` fijando como *listener* el propio `SIPUA`.

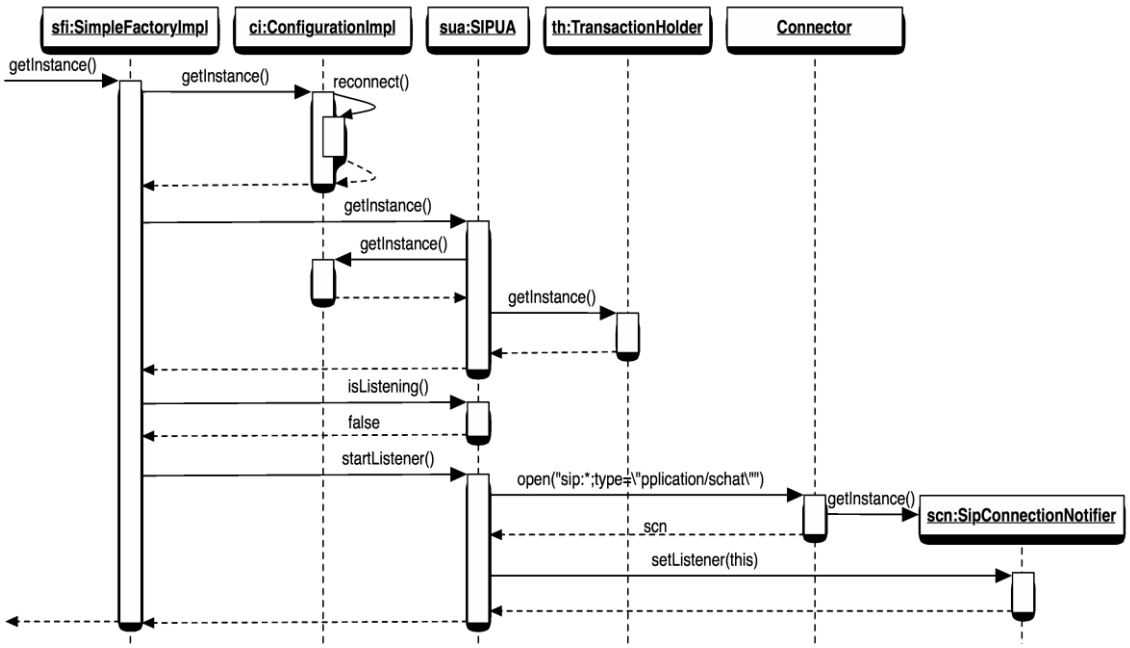


Figura 12: Diagrama de secuencia, obtener la factoría SIMPLE

3.2.5.2. Crear una conexión SIP cliente

La Figura 13 muestra cómo se realiza la creación de las conexiones cliente SIP desde la librería, este método recibe como parámetros obligatorios el `method` que desea crear así como la dirección a donde se desea realizar la conexión, los otros dos parámetros pueden contener datos nulos según el caso. Se procede a una breve explicación:

- A partir del parámetro `addr` se obtiene el `URI` de la dirección de envío y se crea un `SipClientConnection` utilizando la clase `Connector` y el método estático `open()`.
- Se establece como *listener* de la conexión SIP la propia instancia de la clase `SIPUA` y se inicia la petición indicando el `method` deseado y el `SipConnectionNotifier` que recibirá la respuesta.
- Se establecen los credenciales para la conexión en el caso de que la configuración tenga almacenados unos para dicho host.
- Se establecen las cabeceras básicas y las cabeceras recibidas por parámetros.
- Si la conexión tiene cuerpo de mensaje, se establecen las cabeceras del mismo y se rellena el contenido del mensaje.
- Finalmente se crea una transacción realizando una llamada asíncrona que lo inicia.

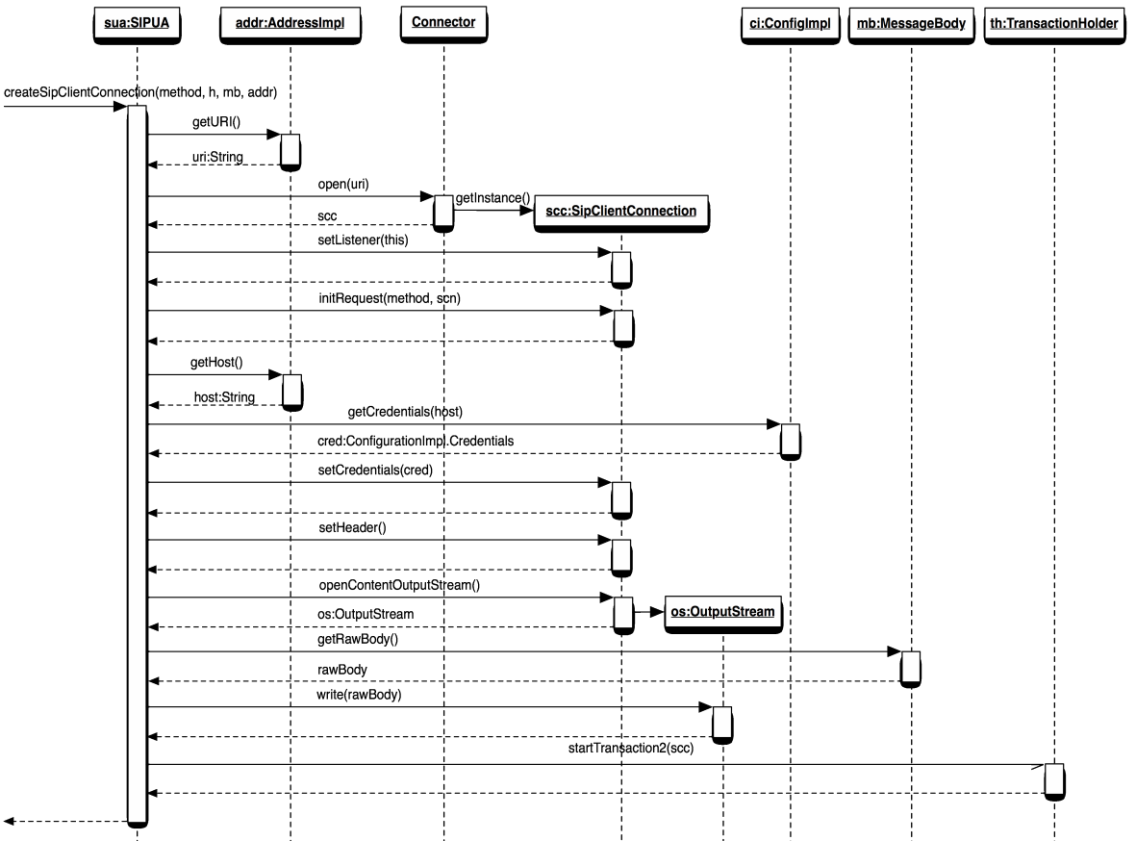


Figura 13: Diagrama de secuencia, crear una conexión SIP Cliente

3.2.5.3. Registrar un agente de usuario

En la Figura 14 se detalla el registro de un determinado agente de usuario, el registro se almacena para su reenvío automático posterior y se almacena la transacción para realizar un seguimiento de la misma. A continuación se detalla el diagrama:

- El agente de usuario recibe una petición `register()` que incluye la dirección que se quiere registrar y se invoca a `sendRegister()` indicando en el parámetro booleano que es una petición de registro.
- Este método incluye las cabeceras por defecto presentes en el agente de usuario e invoca `sendRegisterSipClientConnection()`.
- En este método se incluyen las cabeceras obligatorias `From`, `To` y la cabecera `Expire` con el valor por defecto de la aplicación, después se crea una conexión SIP con el método descrito anteriormente `createSipClientConnection()`.
- Finalmente se crea un nuevo hilo de `Register` que reenvíe el mensaje.

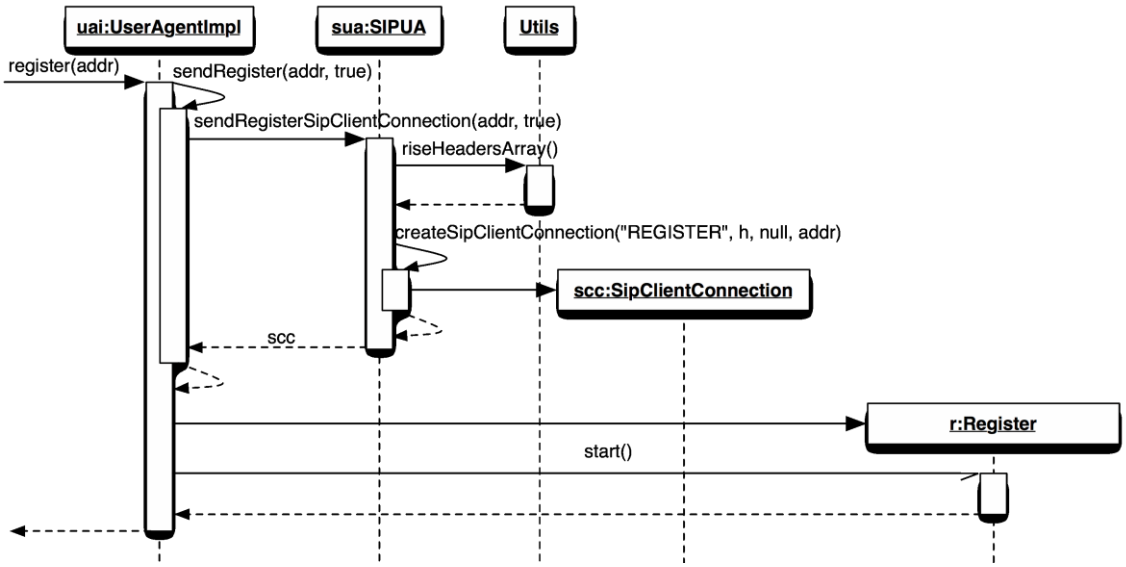


Figura 14: Diagrama de secuencia, registrar un agente de usuario

3.2.5.4. Anular el registro del agente usuario

En la Figura 15 se detalla cómo se anula el registro de un determinado agente de usuario, esta operación también cancela cualquier registro automático que se estuviera realizando. A continuación se detalla el diagrama:

- El agente de usuario recibe una petición `unregister()` que incluye la dirección de la que se quiere finalizar el registro y se invoca a `sendRegister()` indicando en el parámetro booleano que es una petición de anulación de registro.
- Este método incluye las cabeceras por defecto presentes en el agente de usuario e invoca `sendRegisterSipClientConnection()`.
- En este método se incluyen las cabeceras obligatorias `From`, `To` y la cabecera `Expire` con el valor a 0 para indicar que se quiere terminar el registro, después se crea una conexión SIP del mismo modo que en el registro.
- Finalmente se obtiene el hilo `Register` que realiza registros periódicos automáticos y se finaliza el hilo.

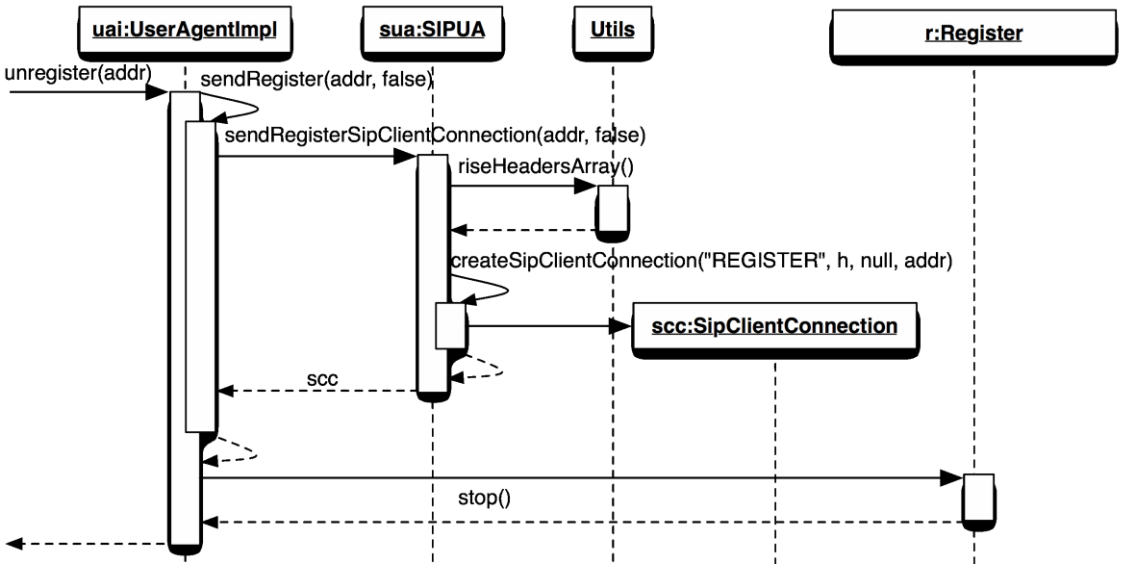


Figura 15: Diagrama de secuencia, anular el registro de un agente de usuario

3.2.5.5. Publicar el estado del usuario

En la Figura 16 se detalla cómo se publica el estado de un determinado agente de usuario. A continuación se detalla el diagrama:

- El `PresenceUserAgent` recibe una petición de publicar el estado con la información de estado almacenada en un documento `PIDFDoc`, de este documento se extrae la dirección del `presentity`, se rellena el cuerpo del mensaje y se invoca a `sendPublishShipClientConnection()` mandando la dirección, el mensaje e indicando en el campo booleano que se desea establecer información presencial.
- En este método se incluyen las cabeceras obligatorias para este tipo de mensaje, `From`, `To`, la cabecera `Expire` con el valor por defecto de la aplicación, la cabecera `Event` se rellena a `presence` y la cabecera `Content-Type` se establece como `application/pidf+xml` después se crea una conexión SIP con el método descrito anteriormente `createSipClientConnection()`.

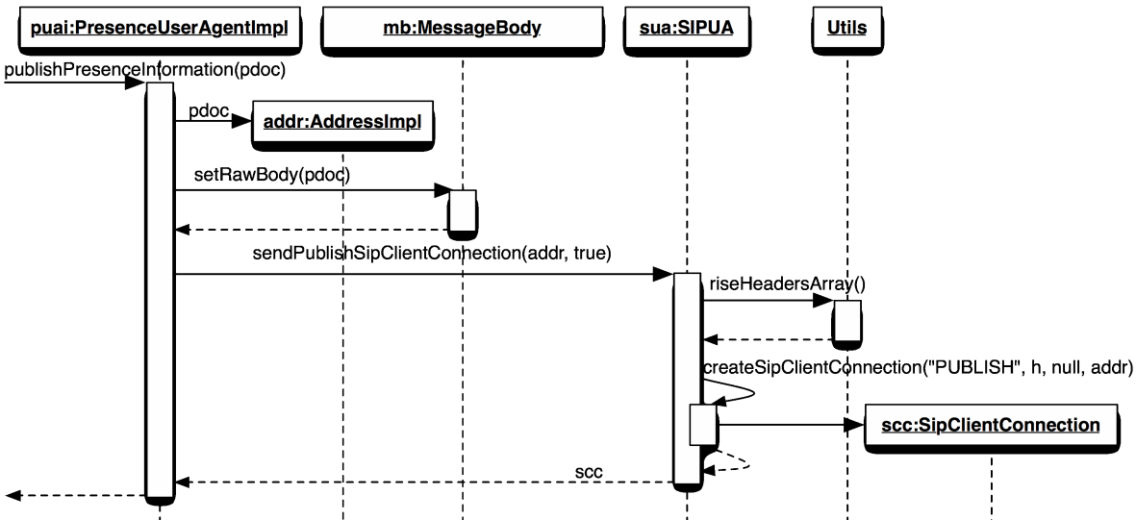


Figura 16: Diagrama de secuencia, publicar el estado del usuario

3.2.5.6. Terminar la publicación

En la Figura 17 se detalla cómo se cancela la publicación de estado lo que implica que el usuario a partir de entonces aparecerá como no conectado. A continuación se detalla el diagrama:

- El `PresenceUserAgent` recibe una petición para terminar la publicación de estado a través del método `terminatePublication()` indicando la dirección para la que se quiere terminar la publicación, e indicando el `eTag` acordado en las publicaciones, se invoca a `sendPublishSipClientconnection()`.
- En este método se incluyen las cabeceras obligatorias para este tipo de mensaje, `From`, `To`, la cabecera `Expire` con el valor a 0 para indicar el fin de publicaciones, la cabecera `Event` se rellena a `presence`, la cabecera `Content-Type` se establece como `application/pidf+xml` y como el `eTag` es distinto de `null` se incluye la cabecera `SIP-If-Match` con el valor del `eTag`; después se crea una conexión SIP con el método descrito anteriormente `createSipClientConnection()`.

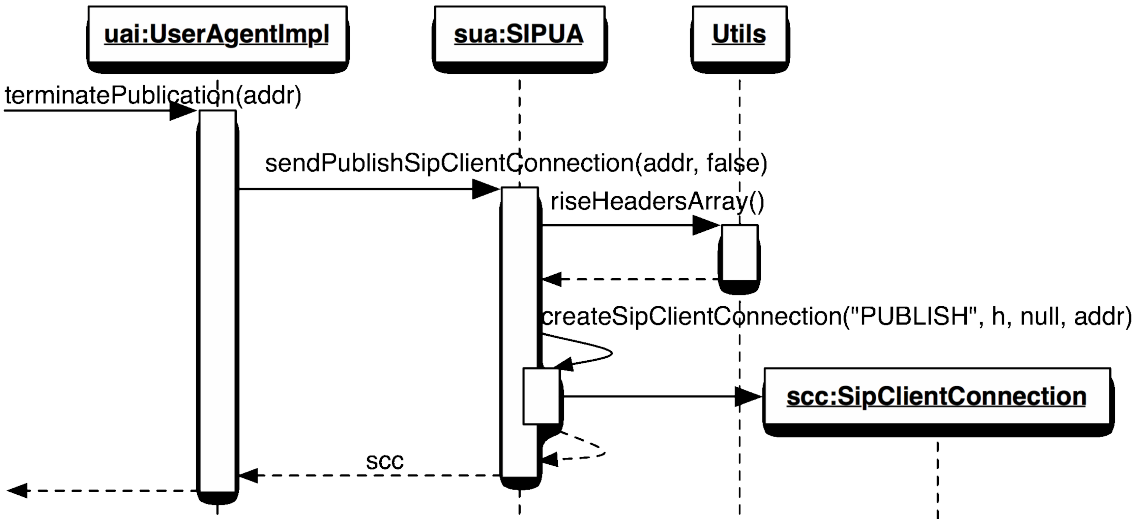


Figura 17: Diagrama de secuencia, terminar la publicación

3.2.5.7. Enviar un mensaje

En la Figura 18 se detalla cómo se envía un mensaje entre dos agentes de usuario. A continuación se detalla el diagrama:

- El `PageModeClientImpl` recibe una petición para enviar un mensaje con una cadena de texto con el contenido del mismo, crea un cuerpo de mensaje con las cabeceras para contener la cadena de texto. Y realiza una petición para enviar el mensaje.
- Se establecen las cabeceras almacenadas por defecto en el agente PageMode, y se invoca a `sendMessageSipClientConnection()`.
- En este método se incluyen las cabeceras obligatorias para este tipo de mensaje, `From`, `To`, después se crea una conexión SIP con el método descrito anteriormente `createSipClientConnection()`.
- Tras esto se crea una nueva transacción que almacena la conexión y se almacena en el `TransactionHolder` para poder llevar un seguimiento de las mismas.

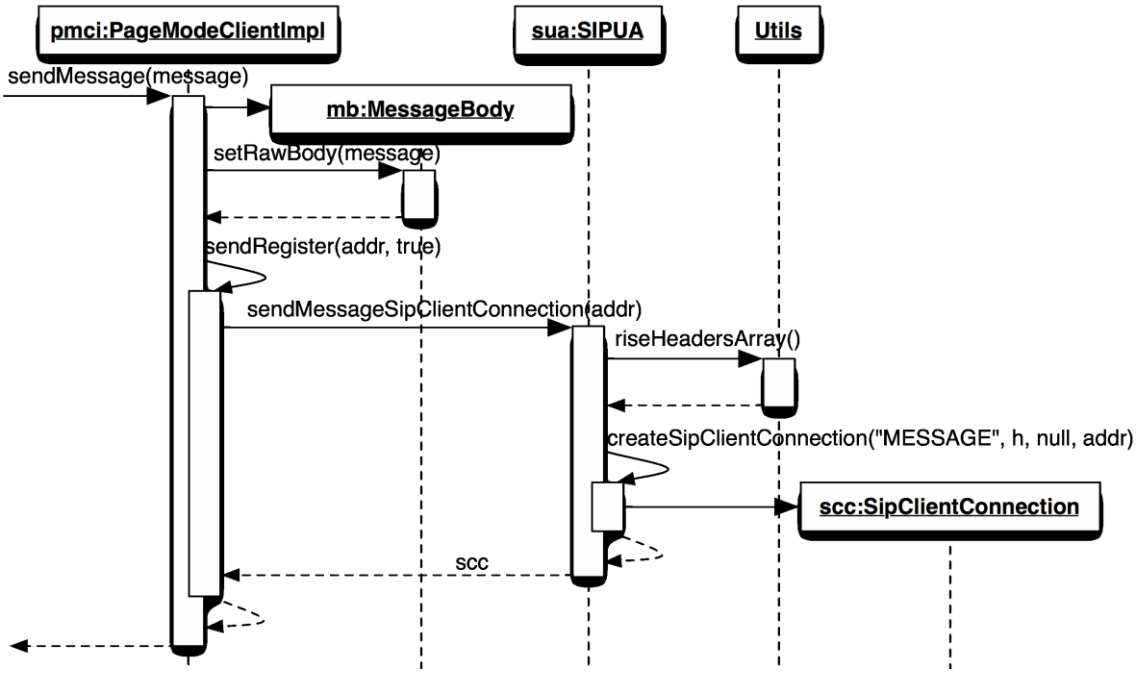


Figura 18: Diagrama de secuencia, enviar un mensaje

3.2.5.8. Recibir un mensaje

En la Figura 19 se muestra cómo se reciben mensajes a través de la librería, a continuación se detallan estos pasos:

- La clase `SIPUA` implementa el *listener* de servidor para conexiones SIP, `SIPServerConnectionListener` por lo que recibe las peticiones enviadas a este servidor. Ante la notificación de una petición se acepta y se crea una transacción para dicha petición.
- Se crea e inicia una transacción para controlar los temporizadores.
- Se obtiene el método de la petición y se comprueba que es del tipo `MESSAGE`, por lo que se obtiene el `PageModeClient` asociado al remitente del mensaje que está almacenado en el `UserAgentWarehouse` y se le notifica la petición.
- Por último se obtiene el Listener asociado a este agente y se le envía la petición.

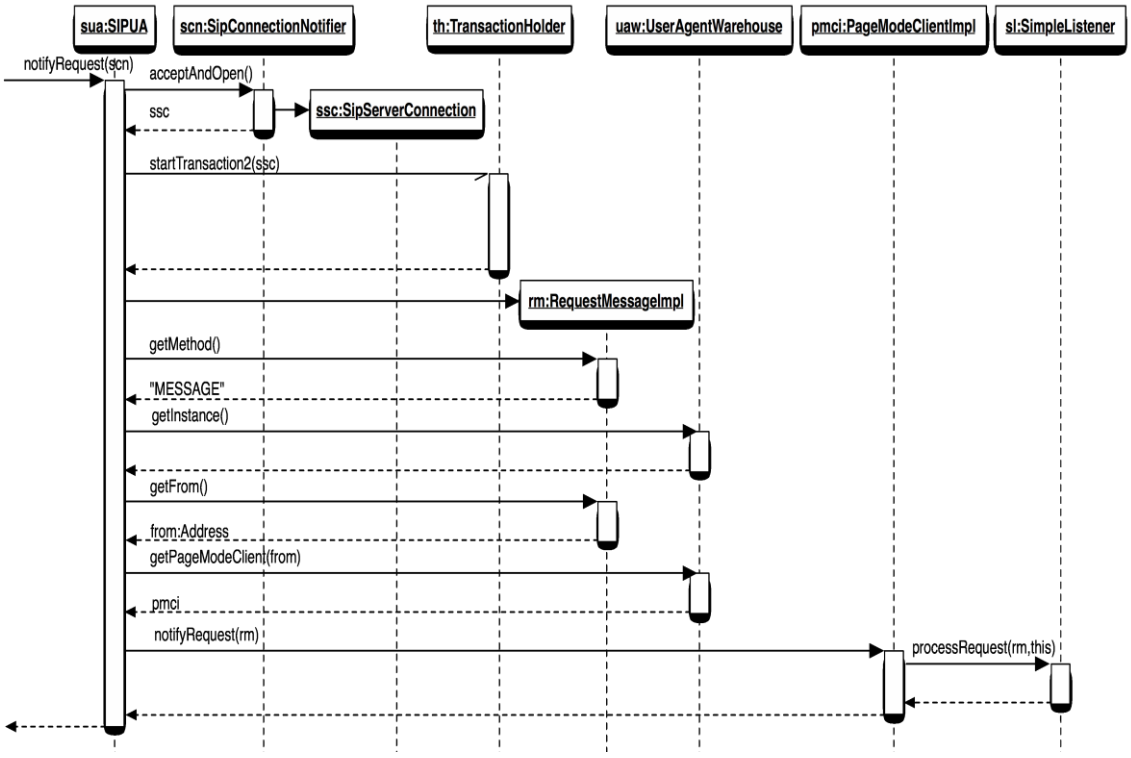


Figura 19: Diagrama de secuencia, recibir un mensaje

3.2.5.9. Suscribirse a un usuario

En la Figura 20 se detalla cómo se realiza una suscripción para recibir la información de presencia de un usuario. A continuación se detalla el diagrama:

- El `WatcherIpl` recibe una petición de para suscribirse a un usuario con la dirección del mismo.
- En este método se incluyen las cabeceras obligatorias para este tipo de mensaje, `From`, `To`, la cabecera `Expire` con el valor por defecto de la aplicación, la cabecera `Event` se rellena a `presence` y la cabecera `Content-Type` se establece como `application/pidf+xml` después se crea una conexión SIP con el método descrito anteriormente `createSipClientConnection()`.

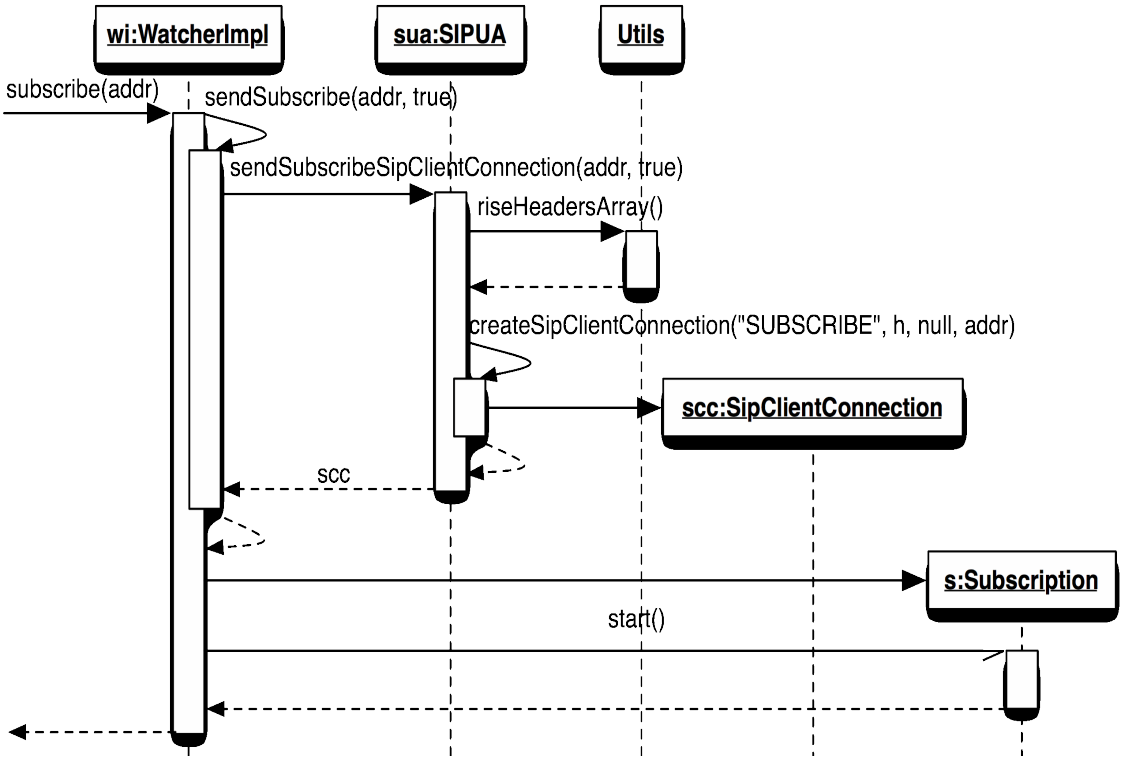


Figura 20: Diagrama de secuencia, suscribirse a un usuario

3.2.5.10. Recibir el estado de un usuario

En la Figura 21 se muestra cómo se reciben una notificación de estado a través de la librería, a continuación se detallan estos pasos:

- La clase `SIPUA` implementa el *listener* de servidor para conexiones SIP, `SIPServerConnectionListener` por lo que recibe las peticiones enviadas a este servidor. Ante la notificación de una petición se acepta y se crea una transacción para dicha petición.
- Se crea una transacción para controlar los temporizadores.
- Se obtiene el método de la petición y se comprueba que es del tipo `NOTIFY`, por lo que se obtiene la instancia del `Watcher` y se le notifica la petición a través del método específico `notifyNotifyRequest()`.
- Por último se obtiene el Listener asociado a este agente y se le envía la petición.

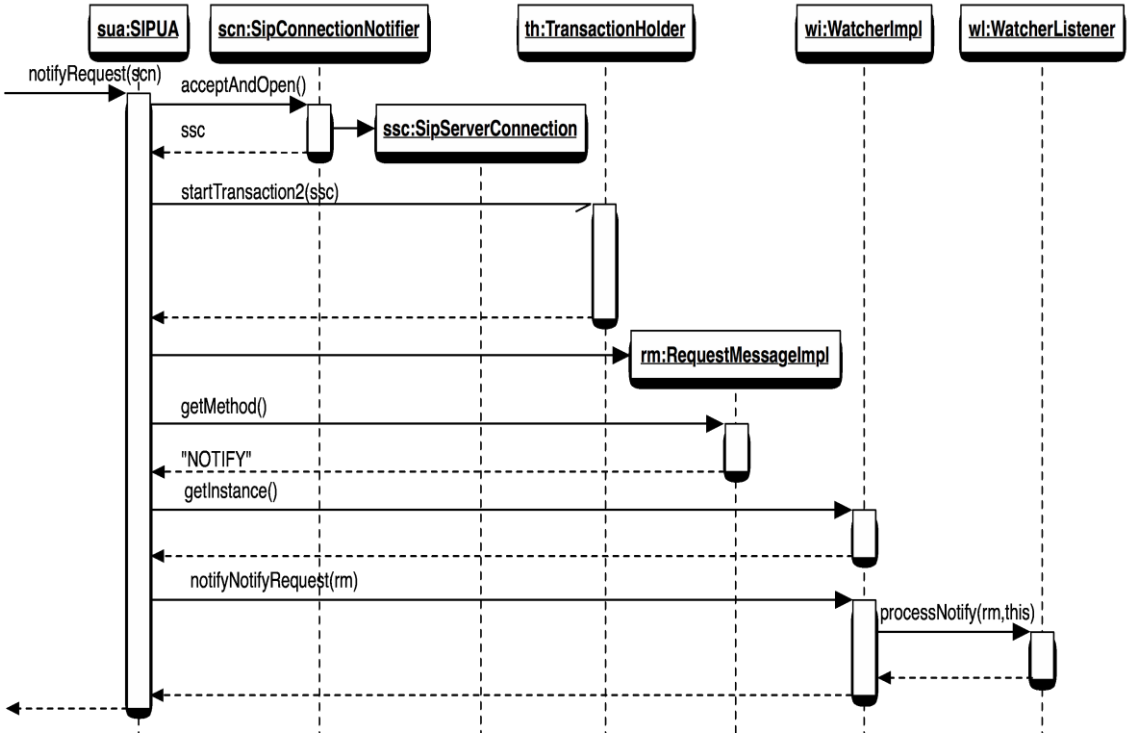


Figura 21: Diagrama de secuencia, recibir el estado de un usuario

3.2.5.11. Anular la suscripción a un usuario

En la Figura 22 se detalla cómo se cancela la suscripción de estado de un usuario, lo que hará que no se reciban mas notificaciones de estado de ese usuario. A continuación se detalla el diagrama:

- El `WatcherImple` recibe una petición para terminar la suscripción de estado a través del método `unsuscribe()` indicando la dirección para la que se quiere terminar la suscripción, si invoca a `sendSubscribe()` indicando que se desea cancelar por medio del parámetro booleano.
- Este método incluye las cabeceras por defecto presentes en el agente de usuario e invoca `sendSubscribeSipClientConnection()`.
- En este método se incluyen las cabeceras obligatorias para este tipo de mensaje, `From`, `To`, la cabecera `Expire` con el valor 0 para realizar la cancelación, la cabecera `Event` se rellena a `presence` y la cabecera `Content-Type` se establece como `application/pidf+xml` después se crea una conexión SIP con el método descrito anteriormente `createSipClientConnection()`.
- Finalmente se obtiene el hilo `Subscription` que realiza actualizaciones de la suscripción periódicas automáticas y se finaliza el hilo.

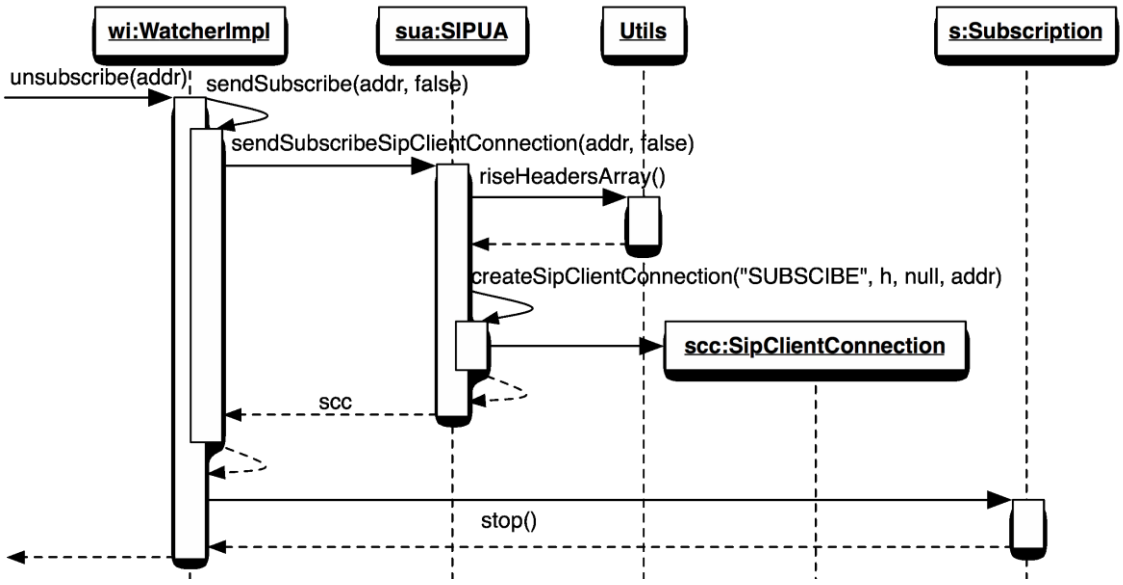


Figura 22: Diagrama de secuencia, anular la subscripción a un usuario

Capítulo 4.

Desarrollo del proyecto SChat

4.1. Análisis

Se especifica el análisis de las necesidades del prototipo SChat, determinando las capacidades que debe tener y los estándares utilizados.

4.1.1. Determinación del alcance del sistema

Se desea crear un prototipo que demuestre la funcionalidad aportada por la librería SIMPLE desarrollada en este proyecto, para ello el prototipo debe tener las siguientes características:

- Permitir gestionar cuentas con direcciones SIP
- Permitir registrar dicha cuenta
- Permitir gestionar los contactos de una cuenta
- Publicar y notificar de los cambios de estado que se produzcan en la cuenta
- Suscribirse y mostrar los cambios de estado de los contactos
- Establecer conversaciones con los contactos

4.1.1.1. Funcionalidades fuera del alcance del sistema

El prototipo SChat no incluirá funcionalidades más allá de las necesarias para demostrar el correcto funcionamiento de la librería.

4.1.2. Especificación de estándares y normas

Se mantienen los mismos estándares y normas que se citan en la sección 3.1.2.

4.1.3. Establecimiento de los requisitos

Especifican las necesidades que el prototipo debe cubrir con un bajo nivel de detalle. Los requisitos se detallan mediante los siguientes atributos:

- **Identificador:** formado por las siglas de Requisito y el tipo del mismo seguido de un número secuencial dado por orden de creación del requisito.
- **Descripción:** explicación formal del propósito del requisito.
- **Prioridad:** si el software se desarrolla en múltiples etapas indica la importancia de cubrir unos requisitos antes de otros
  - **Alta:** el requisito debe solucionarse en fases iníciales del desarrollo software.
  - **Media:** se debe solucionar en las fases medias del desarrollo del software.
  - **Baja:** se soluciona el requisito al final del ciclo de desarrollo.
- **Estabilidad:** mide la posibilidad de cambios durante el ciclo de vida del proyecto
  - **Alta:** no va a sufrir cambios.
  - **Media:** se esperan un número de cambios mínimos y no críticos o ninguno.
  - **Baja:** es posible que cambien mucho o que el cambio sea crítico en el sistema.
- **Origen:** Determina la fuente de la que proviene el requisito.
- **Grado de Verificación, (G. Verificación):** mide si se puede comprobar el cumplimiento del requisito.
  - **Alta:** el requisito es fácil de verificar.
  - **Media:** el requisito se puede verificar pero implica numerosas pruebas.
  - **Baja:** el número de pruebas que implica es demasiado elevado o el cumplimiento del requisito es subjetivo de los usuarios.

4.1.3.1. Requisitos de Usuario

A continuación se detallan los requisitos de usuario que se han establecido para este proyecto.

Tabla 35: RU01

Identificador: RU01			
Descripción: El sistema debe gestionar cuentas con direcciones SIP			
Prioridad	Media	**Estabilidad**	Media
Origen	Librería	**G. Verificación**	Alta

Tabla 36: RU02

Identificador: RU02			
Descripción: El sistema debe gestionar el registro de una cuenta en un servidor de registro o *registrar*			
Prioridad	Alta	**Estabilidad**	Alta
Origen	Librería	**G. Verificación**	Alta

Tabla 37: RU03

Identificador: RU03			
Descripción: El sistema debe gestionar los contactos de una cuenta			
Prioridad	Media	**Estabilidad**	Media
Origen	Librería	**G. Verificación**	Alta

Tabla 38: RU04

Identificador: RU04			
Descripción: El sistema debe gestionar la información de presencia			
Prioridad	Alta	**Estabilidad**	Alta
Origen	Librería	**G. Verificación**	Alta

Tabla 39: RU05

Identificador: RU05			
Descripción: El sistema debe poder realizar mensajería instantánea con los contactos			
Prioridad	Alta	**Estabilidad**	Alta
Origen	Librería	**G. Verificación**	Alta

4.1.4. Especificación de casos de uso

A continuación se muestran los casos de uso en función del tipo de acceso y de los permisos de los que disponga el usuario:

4.1.4.1. Gestión de cuentas

Como se muestra en la Figura 23 el usuario al entrar en la aplicación debe poder crear cuentas de usuario, conectarse con alguna de las cuentas que ya estén creadas o borrar alguna cuenta. El *feedback* hacia el usuario dentro de estas operaciones es la aparición o desaparición de la cuenta de usuario de la pantalla en el caso de la creación o el borrado y de un cambio de pantalla hacia la gestión de contactos de la cuenta que se haya conectado.

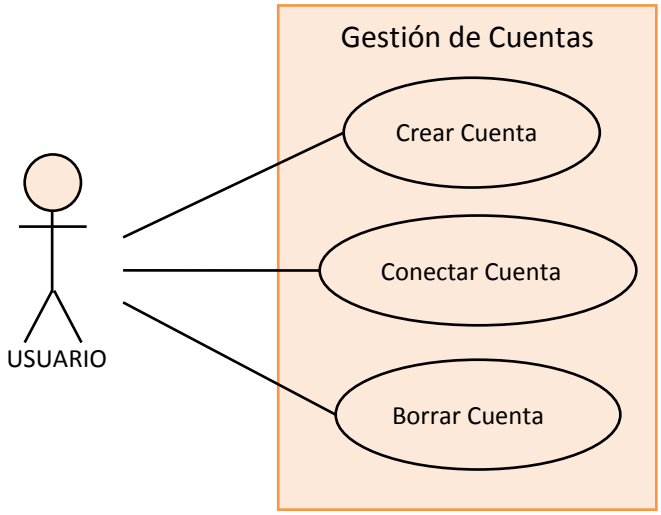


Figura 23: Diagrama de casos de uso, gestión de cuentas

4.1.4.2. Gestión de contactos

Como se muestra en la Figura 24 los usuarios pueden crear o borrar grupos para los contactos, crear o borrar contactos, seleccionar un contacto para abrir su ventana de chat o desconectar su cuenta. El *feedback* en la creación y el borrado tanto de los grupos como de los contactos será la aparición o desaparición de dicho contacto o grupo de la interfaz, en el caso de que el usuario inicie una conversación cambiará de pantalla y si desconecta la cuenta volverá a la gestión de cuentas.

En todo momento el usuario recibirá los cambios de estado de sus contactos y se le mostrará por la pantalla a través del sistema y nada mas conectarse el propio sistema publicará su cambio de estado como conectado.

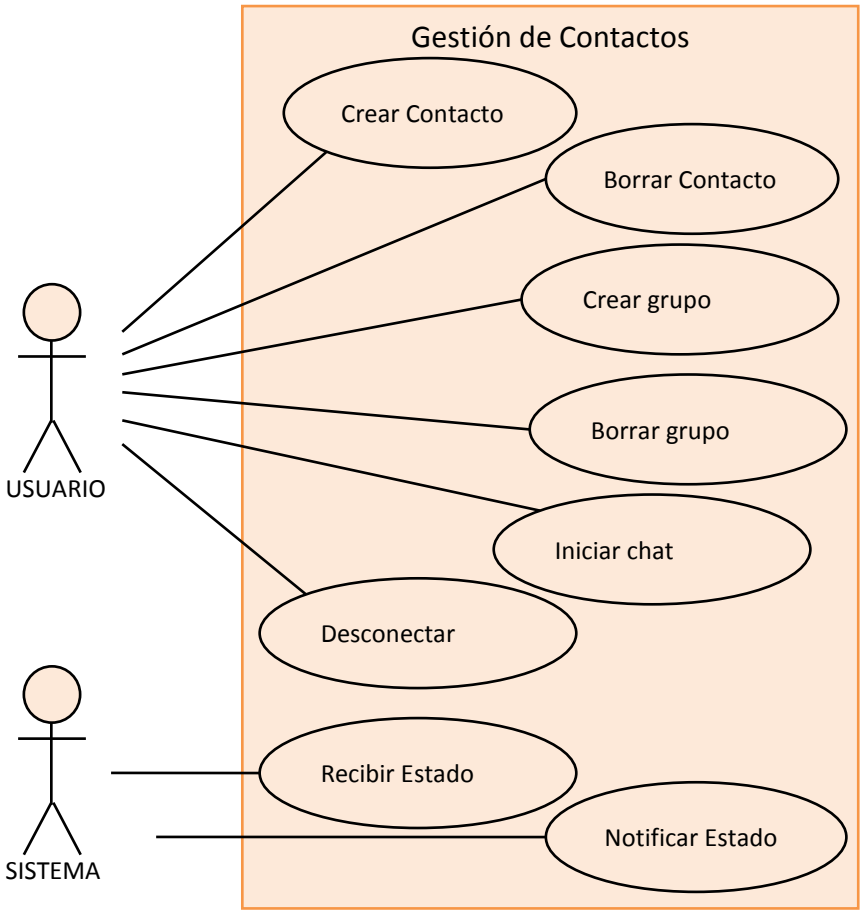


Figura 24: Diagrama de casos de uso, gestión de contactos

4.1.4.3. Interfaz de chat

Como se muestra en la Figura 25 el usuario puede enviar mensajes y visualizar los mensajes que ha recibido el sistema.

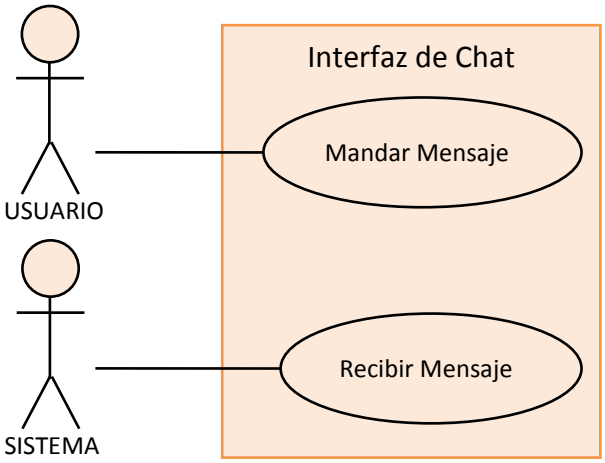


Figura 25: Diagrama de casos de uso, interfaz de chat

4.1.5. Establecimiento de requisitos software

Los requisitos software detallan la especificación completa del sistema a desarrollar, precisando su funcionalidad y sus restricciones de diseño y funcionamiento.

4.1.5.1. Requisitos funcionales

Tabla 40: RFunc01

Identificador: RFunco1			
Descripción: El sistema debe poder crear una cuenta de usuario a partir de una dirección SIP			
Prioridad	Media	**Estabilidad**	Media
Origen	RU01	**G. Verificación**	Alta

Tabla 41: RFunc02

Identificador: RFunco2			
Descripción: El sistema debe poder borrar una cuenta de usuario			
Prioridad	Media	**Estabilidad**	Media
Origen	RU01	**G. Verificación**	Alta

Tabla 42: RFunc03

Identificador: RFunco3			
Descripción: El sistema debe poder conectarse a una cuenta de usuario, al conectarse se debe registrar la dirección SIP asociada a la cuenta en un servidor de registro o *registrar*.			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU02	**G. Verificación**	Alta

Tabla 43: RFunc04

Identificador: RFunco4			
Descripción: El sistema debe poder crear un grupo de usuarios			
Prioridad	Media	**Estabilidad**	Media
Origen	RU03	**G. Verificación**	Alta

Tabla 44: RFunc05

Identificador: RFunco5			
Descripción: El sistema debe poder borrar un grupo de usuarios			
Prioridad	Media	**Estabilidad**	Media
Origen	RU03	**G. Verificación**	Alta

Tabla 45: RFunc06

Identificador: RFunco6			
Descripción: El sistema debe poder crear un contacto que consiste en una dirección SIP y poder asociarla a uno de los grupos que existan para la cuenta si se desea.			
Prioridad	Media	**Estabilidad**	Media
Origen	RU03	**G. Verificación**	Alta

Tabla 46: RFunc07

Identificador: RFunco7			
Descripción: El sistema debe poder borrar un contacto			
Prioridad	Media	**Estabilidad**	Media
Origen	RU03	**G. Verificación**	Alta

Tabla 47: RFunc08

Identificador: RFunco8			
Descripción: El sistema debe suscribirse a los cambios de estado de sus contactos			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU04	**G. Verificación**	Alta

Tabla 48: RFunc09

Identificador: RFunco9			
Descripción: El sistema debe publicar su estado al conectarse			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU04	**G. Verificación**	Alta

Tabla 49: RFunc10

Identificador: RFunc10			
Descripción: El sistema debe cancelar su publicación de estado al desconectarse			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU04	**G. Verificación**	Alta

Tabla 50: RFunc11

Identificador: RFunc11			
Descripción: El sistema debe cancelar todas las suscripciones de estado al desconectarse			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU04	**G. Verificación**	Alta

Tabla 51: RFunc12

Identificador: RFunc12			
Descripción: El sistema debe poder enviar mensajes a un contacto			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU05	**G. Verificación**	Alta

Tabla 52: RFunc13

Identificador: RFunc13			
Descripción: El sistema debe poder recibir mensaje de un contacto			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU05	**G. Verificación**	Alta

Tabla 53: RFunc14

Identificador: RFunc14			
Descripción: El sistema debe poder desconectar una cuenta de usuario, al desconectar se debe anular el registro de la dirección SIP en el servidor de registro o *registrar*.			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU02	**G. Verificación**	Alta

4.1.5.2. Requisitos de interfaz

Tabla 54: RInterf01

Identificador: Rinterfo1			
Descripción: El sistema debe notificar los cambios de estado de un usuario, para ello se utilizará un icono verde para indicar que el usuario está conectado y un icono gris para indicar que está desconectado			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU04	**G. Verificación**	Alta

Tabla 55: Rinterf02

Identificador: Rinterfo2			
Descripción: El sistema diferenciará los mensajes escritos por un contacto con su dirección SIP y los escritos por el usuario con una referencia a el mismo. Ejm. "YO"			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU04	**G. Verificación**	Alta

4.1.5.3. Requisitos de verificación

Tabla 56: Rverif01

Identificador: RVerifo1			
Descripción: El sistema debe comprobar el formato de las direcciones SIP introducidas al crear cuentas o contactos mostrando un mensaje de error en caso contrario			
Prioridad	Alta	**Estabilidad**	Alta
Origen	RU01, RU03	**G. Verificación**	Alta

4.1.6. Matriz de trazabilidad de requisitos software

Requisitos software	**Requisitos de Usuario**				
	RU01	**RU02**	**RU03**	**RU04**	**RU05**
RFunc01	X				
RFunc02	X				
RFunc03		X			
RFunc04			X		
RFunc05			X		
RFunc06			X		
RFunc07			X		
RFunc08				X	
RFunc09				X	
RFunc10				X	
RFunc11				X	
RFunc12					X
RFunc13					X
RFunc14		X			
RInterf01				X	
RInterf02				X	
RVerif01	X		X		

4.2. Diseño

Se especifica el diseño de la aplicación, la arquitectura, clases y el diseño detallado.

4.2.1. Tecnologías utilizadas

Se utilizan las mismas tecnologías especificadas en la sección 3.2.1 y se añade la librería implementada para la definición de los JSR 164 y JSR 165.

4.2.2. Arquitectura

La arquitectura utilizada en la aplicación SChat corresponde a un patrón Modelo Vista Controlador (MVC), de esta forma se consigue la separación de los datos, la lógica de control y el interfaz de la aplicación.

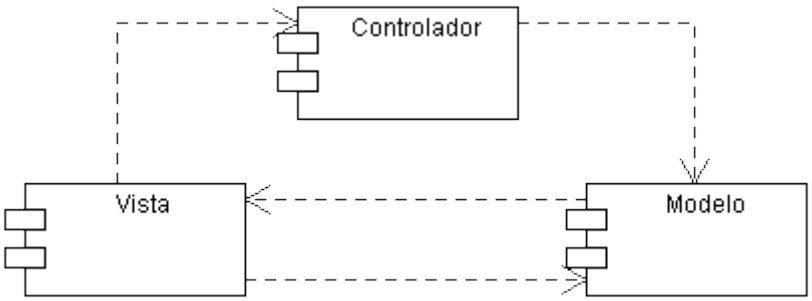


Figura 26: Arquitectura de SChat

- **Modelo:** Almacena toda la información del sistema de manera persistente asegurando la integridad de todos los datos. El almacenamiento de datos se realiza mediante `RecordStore` creándose una estructura de clases para el tratamiento y control de los mismos, esta estructura se explicará en los detalles del diseño, dentro del modelo se recoge la clase `SIMPLEUA` que implementaría un agente de usuario con la funcionalidad básica para el protocolo SIMPLE.
- **Vista:** Proporciona un modo de representación de los datos utilizando las primitivas de interfaz aportadas por `javax.microedition.lcdui`.
- **Controlador:** Proporciona la lógica del sistema, recibiendo las entradas desde la vista y manejando los eventos, se encarga de invocar cambios en el modelo y de enviar las respuestas apropiadas a la vista. Permite la ejecución de los métodos necesarios para todo el manejo del sistema y es el responsable de la integridad de los datos de la aplicación. El controlador está aportado por la propia estructura de funcionamiento de eventos de J2ME

4.2.3. Pautas de diseño

Se mantienen las mismas pautas de diseño que en el proyecto anterior, se pueden consultar en la sección 3.2.3

4.2.4. Diseño detallado

La Figura 27 muestra el diagrama de clases del proyecto SChat en una vista global del sistema incluyendo las relaciones entre las clases. Posteriormente se mostrará la especificación de los subsistemas vista y modelo.

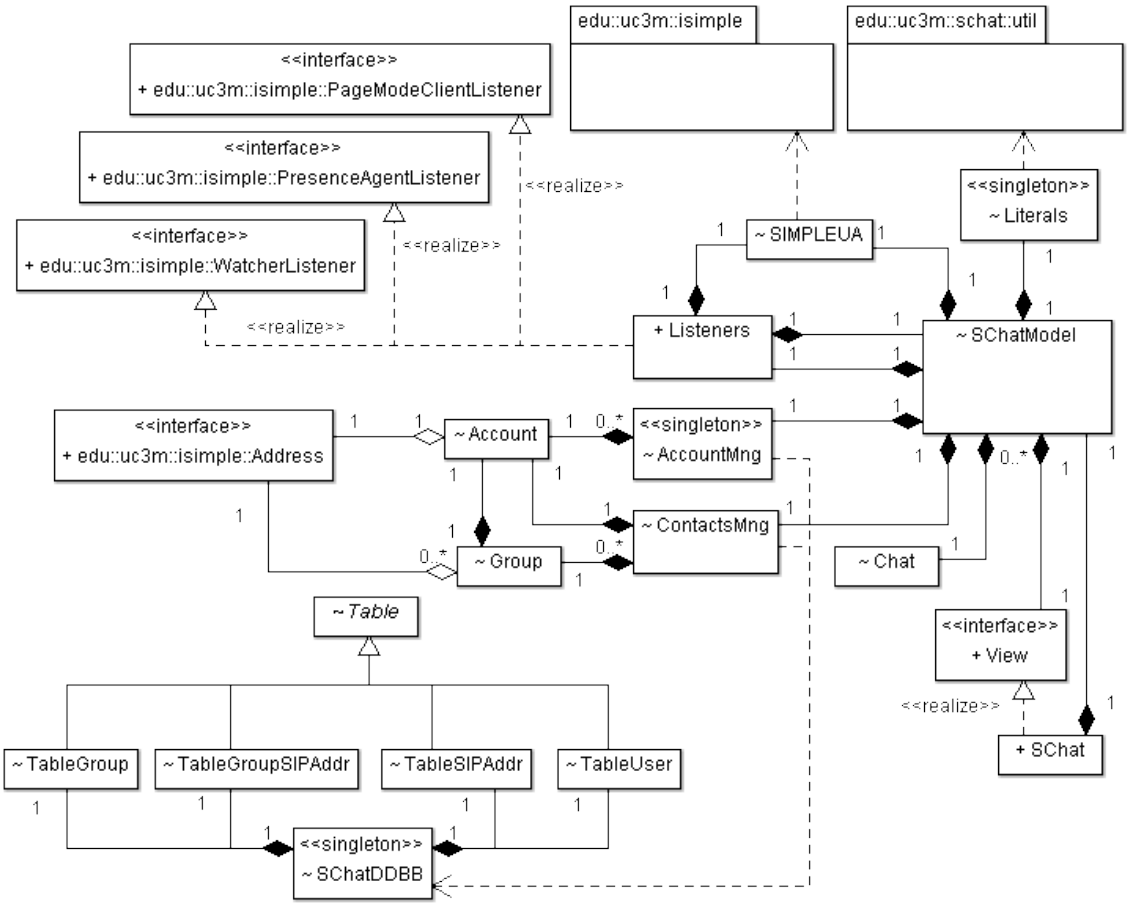


Figura 27: Diagrama de clases de SChat

4.2.4.1. Subsistema Vista

El subsistema vista se encarga de la representación de la información aportada por el modelo, esta representación se define en la clase `SChat`, en la Figura 28 se puede ver que la clase `SChat` tiene una instancia de `SChatModel` a la que accede directamente a través de sus métodos públicos para obtener la información requerida y emitir los comandos que se reciben a través del controlador. Por otro lado el modelo también posee una instancia de `SChat` pero únicamente tiene acceso a ella a través de la interfaz `View` permitiéndole notificar de manera proactiva mensajes o cambios producidos internamente en el modelo. Finalmente se observa que la obtención de los literales de la interfaz está preparada para la internacionalización de los mismos por medio de ficheros *locale*.

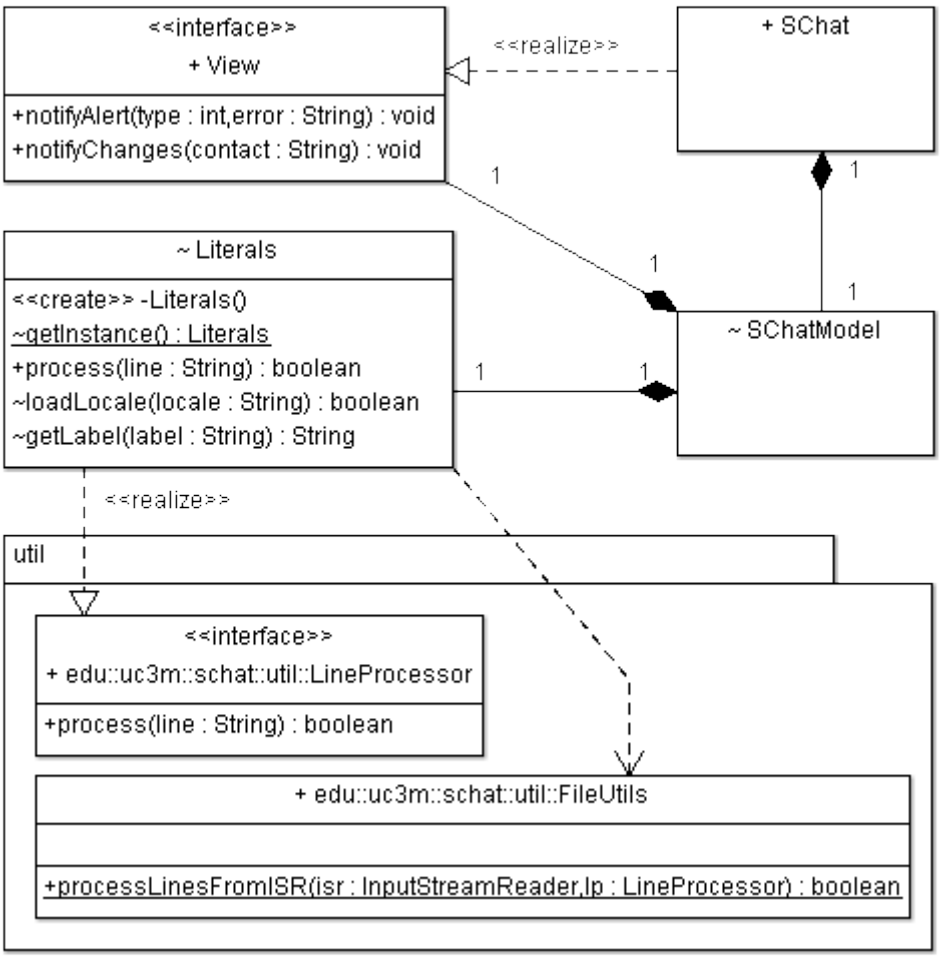


Figura 28: Diagrama de clases del subsistema vista

La Figura 29 muestra las transiciones que se realizan entre los diferentes menús del proyecto SChat, se observa que existen seis páginas que ofrecen distinta funcionalidad:

- **WelcomePage:** La página principal muestra las distintas cuentas SIP creadas en la aplicación, permite crear, borrar o conectar una cuenta; también es el punto de salida del programa.
- **AddAccountPage:** esta página muestra un formulario para rellenar el usuario, el reino y el password del usuario para el reino y asociarlos a la cuenta.
- **ContactsPage:** muestra los contactos por grupos que tiene el usuario, al lado de cada usuario aparece un icono que indica la información presencial del mismo (el icono es gris si el usuario está desconectado y verde en caso contrario). Se permite crear y borrar tanto grupos como contactos, iniciar chats y desconectar la cuenta.
- **AddGroupPage:** pide el nombre que tendrá el grupo.
- **AddContactPage:** para crear un contacto se debe especificar el usuario, el reino al que pertenece y el grupo en el que se quiere almacenar.
- **ChatPage:** muestra los mensajes mandados y recibidos de la conversación

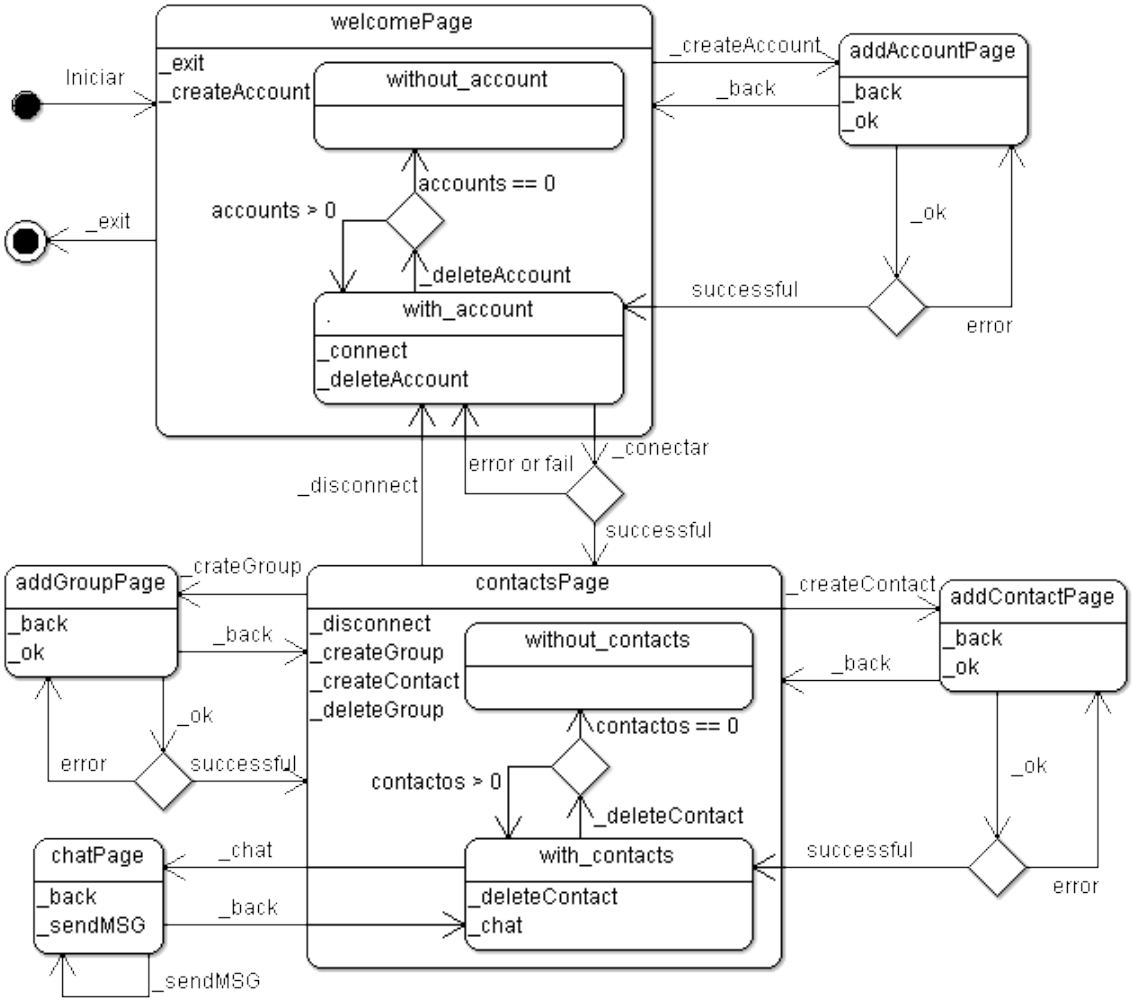


Figura 29: Transiciones en los menús de la interfaz SChat

4.2.4.2. Subsistema Modelo

El modelo implementa dos paquetes de funcionalidades, por un lado se implementa un modelo de persistencia para las cuentas, los contactos y los grupos; por otro lado se implementa un modelo de comunicación SIMPLE para el intercambio de mensajes y la gestión de información de presencia.

Modelo de persistencia

El modelo de persistencia tiene que almacenar información de las cuentas, contactos y grupos, se ha seguido como metodología realizar un diseño relacional conceptual desde el que se ha realizado una implementación basada en `RecordStore` de J2ME.

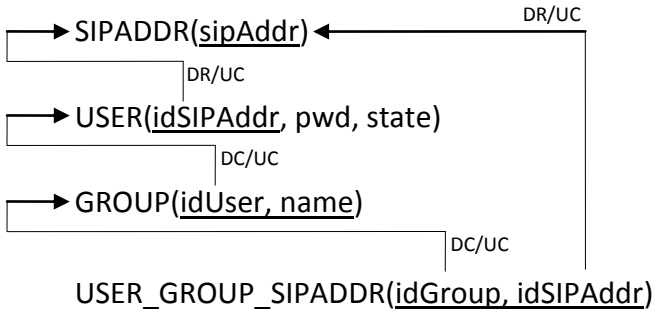


Figura 30: Diagrama relacional del modelo de persistencia.

Ante la ausencia de una base de datos relacional en los terminales móviles se desarrolla una estructura de clases Java apoyada en `RecordStore`, se crea una clase abstracta `Table` que representa una tabla y corresponde a un `RecordStore` específico, en esta clase se definen los métodos genéricos para la apertura y cierre de la tabla, así como para la inserción, recuperación y modificación de los datos.

Cada una de las clases hijas de `Table` define una de las tablas indicadas en el diseño relacional de la Figura 30. Estas clases implementan los métodos concretos de gestión de las tuplas de la tabla.

El acceso a los datos de persistencia se realiza por medio de la clase `SChatDDBB` que implementa el patrón *singleton*, esta clase mapea el modelo relacional de tablas con el modelo de clases del sistema.

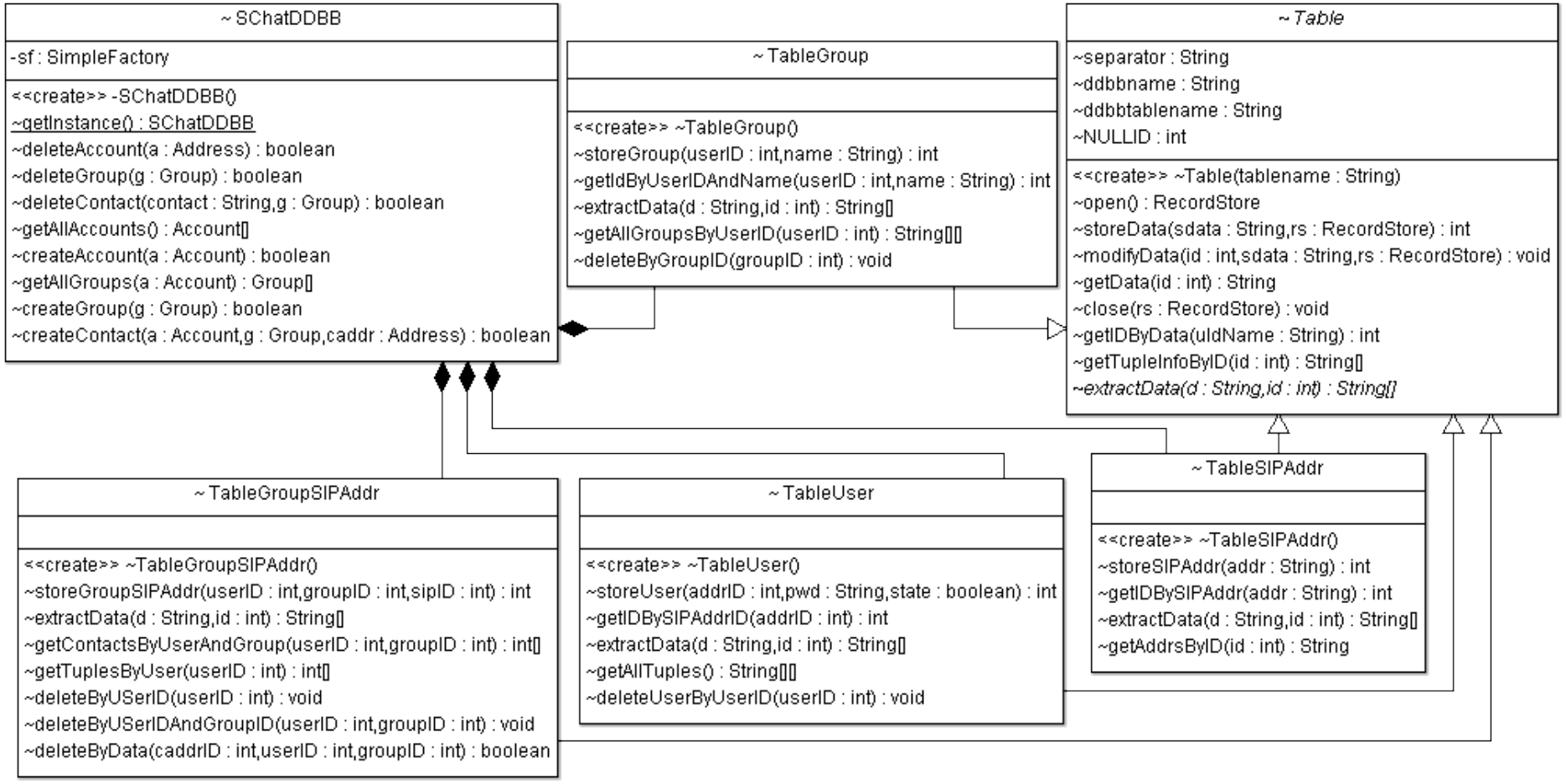


Figura 31: Diseño de clases del modelo de persistencia.

Modelo de comunicación SIMPLE

El modelo de comunicación SIMPLE se encarga de la transmisión de mensajes utilizando la librería SIMPLE diseñada, se divide en dos clases principales:

- **Listeners:** esta clase realiza los tres *listeners* definidos en la librería, a través de ella se recibe información de otros agentes de usuario que tengan como destino la aplicación final, esta información puede ser información presencial, mensajes de mensajería instantánea o respuestas SIP/SIMPLE según el caso el *listener* notificará al modelo para que actualice el estado de un contacto, añada el mensaje a la conversación entre el usuario y el contacto o muestre un mensaje de aviso con el contenido de la respuesta SIP/SIMPLE.
- **SIMPLEUA:** esta clase define el agente de usuario SIMPLE para una cuenta, indica el estado de la conexión, y define las operaciones necesarias para conectar y desconectar un cliente como para suscribirse, mandar mensajes y anular las subscripciones a un contacto.

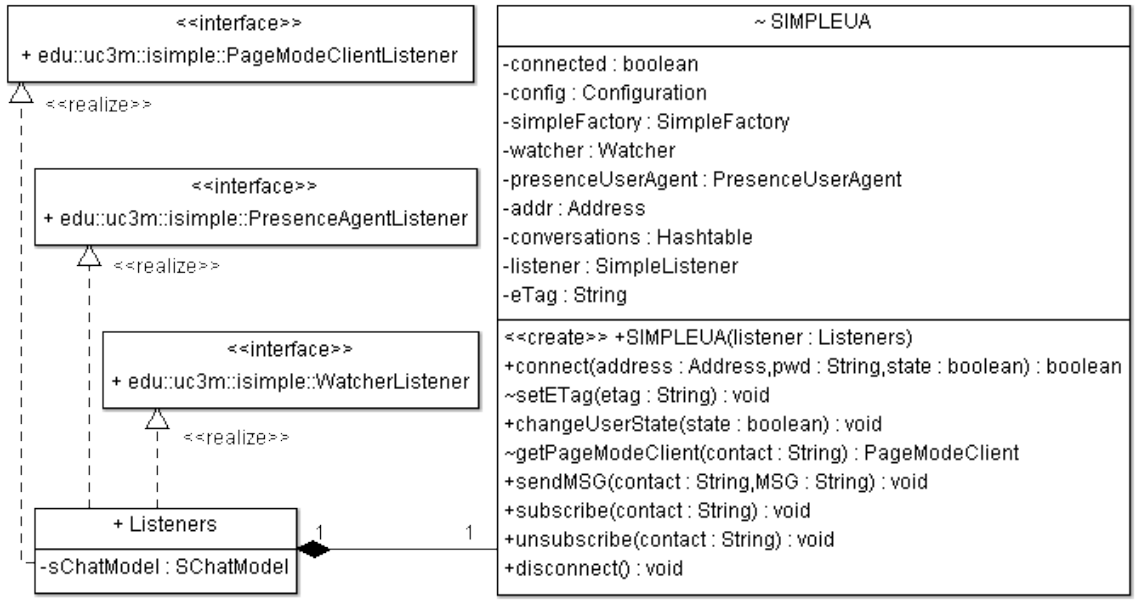


Figura 32: Diseño de clases del modelo de comunicación SIMPLE.

4.2.5. Diseño de Interfaces

La aplicación SChat comienza con una pantalla de bienvenida, si la aplicación no tiene cuentas almacenadas da un mensaje de aviso y ofrece como únicas opciones salir o crear una cuenta nueva; en el caso de que existan cuentas de usuario muestra una lista seleccionable de las mismas y amplía las opciones disponibles permitiendo borrar la cuenta o conectarse a la misma. Para crear una cuenta se debe incluir el usuario el reino y el password asociado a ese usuario para ese reino.

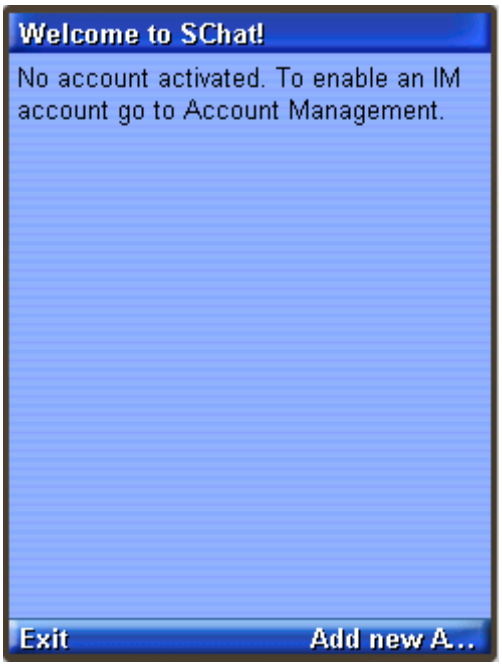


Figura 33: Pantalla inicial sin cuentas

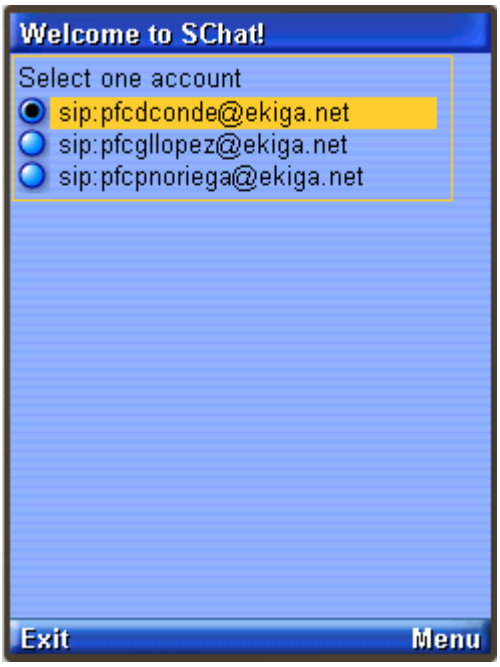


Figura 34: Pantalla inicial con cuentas

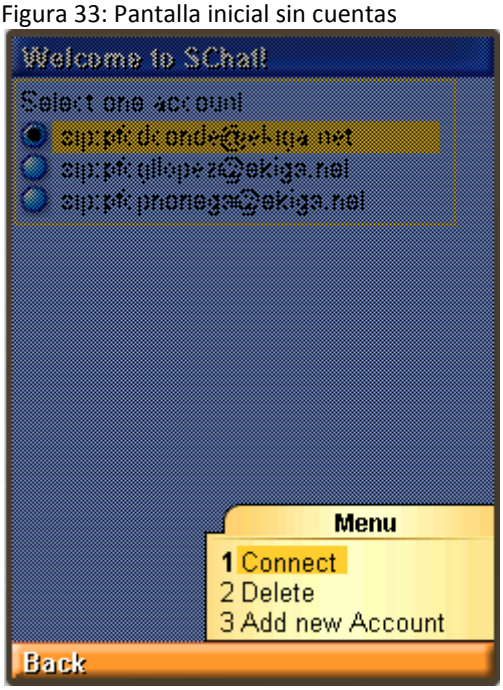


Figura 35: Menú de la pantalla inicial

Figura 36: Pantalla de añadir cuenta

Tras conectarse a una cuenta aparecen los contactos ordenados por grupos de la cuenta, en esta pantalla se puede ver el estado de los contactos representándose con un cuadrado gris los usuarios desconectados y con uno verde los usuarios conectados. Se ofrece un menú en el que se puede iniciar una conversación, añadir o borrar contactos o grupos y desconectar la cuenta. Para crear un grupo se debe aportar el nombre del mismo y en el caso de los contactos se debe indicar el usuario el reino y a qué grupo se desea asignar.

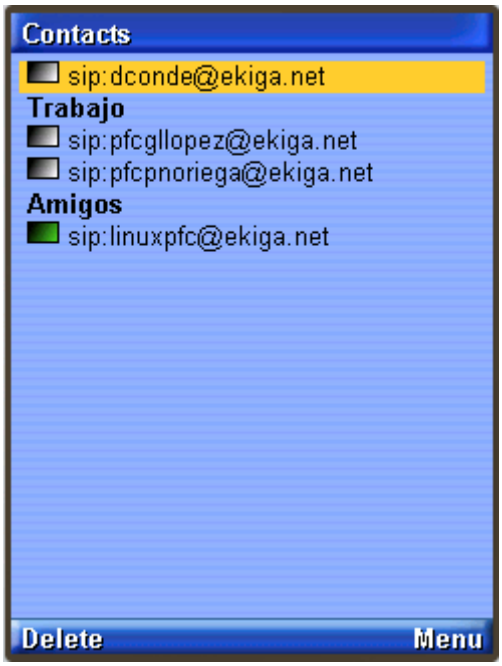


Figura 37: Pantalla de contactos

Figura 38: Menú de la pantalla de contactos

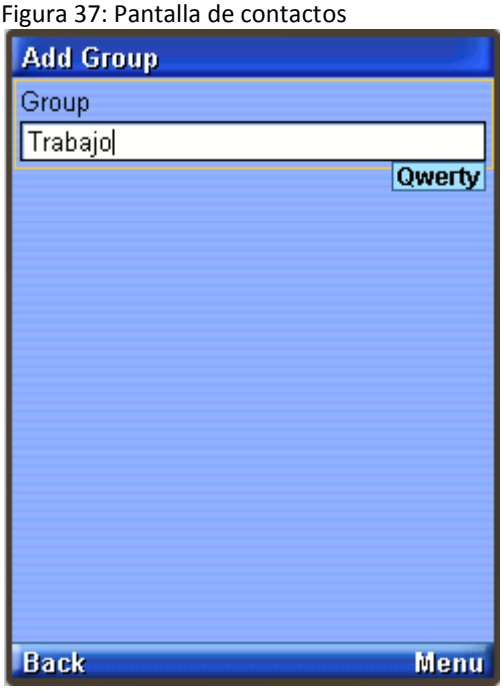


Figura 39: Pantalla de añadir grupo

Figura 40: Pantalla de añadir contacto

En la pantalla de chat se muestra la conversación entre los usuarios, en la primera línea se indica con qué usuario se está manteniendo la conversación. Después se muestra un identificador en negrita del interlocutor con su discurso; tras la conversación se muestra una caja de texto en la que se introduce el texto que se desea enviar.

Figura 41: Pantalla de conversación

Capítulo 5.

Pruebas y validación

En este capítulo se muestra la validación de la aplicación SChat. En esta validación se busca comprobar que se cumplen los siguientes objetivos:

- **Objetivo A:** La aplicación cumple con todos los requisitos de usuario y de software definidos.
- **Objetivo B:** La librería es consistente y ofrece toda la funcionalidad para la que ha sido diseñada

Para la realización de las pruebas se ha utilizado el servicio de registro SIP ekiga.net en el que se han creado 3 cuentas de usuario distintas con la siguiente configuración:

Tabla 57: Datos de cuentas de usuario en el servidor de registro ekiga.net

Username	SIP Address	Password
linuxpfc	sip:linuxpfc@ekiga.net	Linux_pfc0
dconde	sip:dconde@ekiga.net	pfc.dconde
pfcdconde	sip:pfcdconde@ekiga.net	pfcdc0nd3

Durante la realización de las pruebas se han utilizado dos máquinas:

- **PC de desarrollo:** en el que se ejecuta la simulación.
- **PC de ayuda:** que ejecuta el cliente SIP communication.

5.1. Especificación del entorno de pruebas

El proyecto consta de dos entornos de pruebas, uno de simulación y otro en terminales reales.

- **Entorno de simulación:** durante las pruebas de simulación se utiliza el entorno de desarrollo con un SDK instalado para los dispositivos móviles. Este entorno permite la ejecución y depuración sin la necesidad de utilizar ningún dispositivo móvil, en el desarrollo de este proyecto se ha utilizado NetBeans 6.8 con la configuración de simulación especificada en la Figura 42.

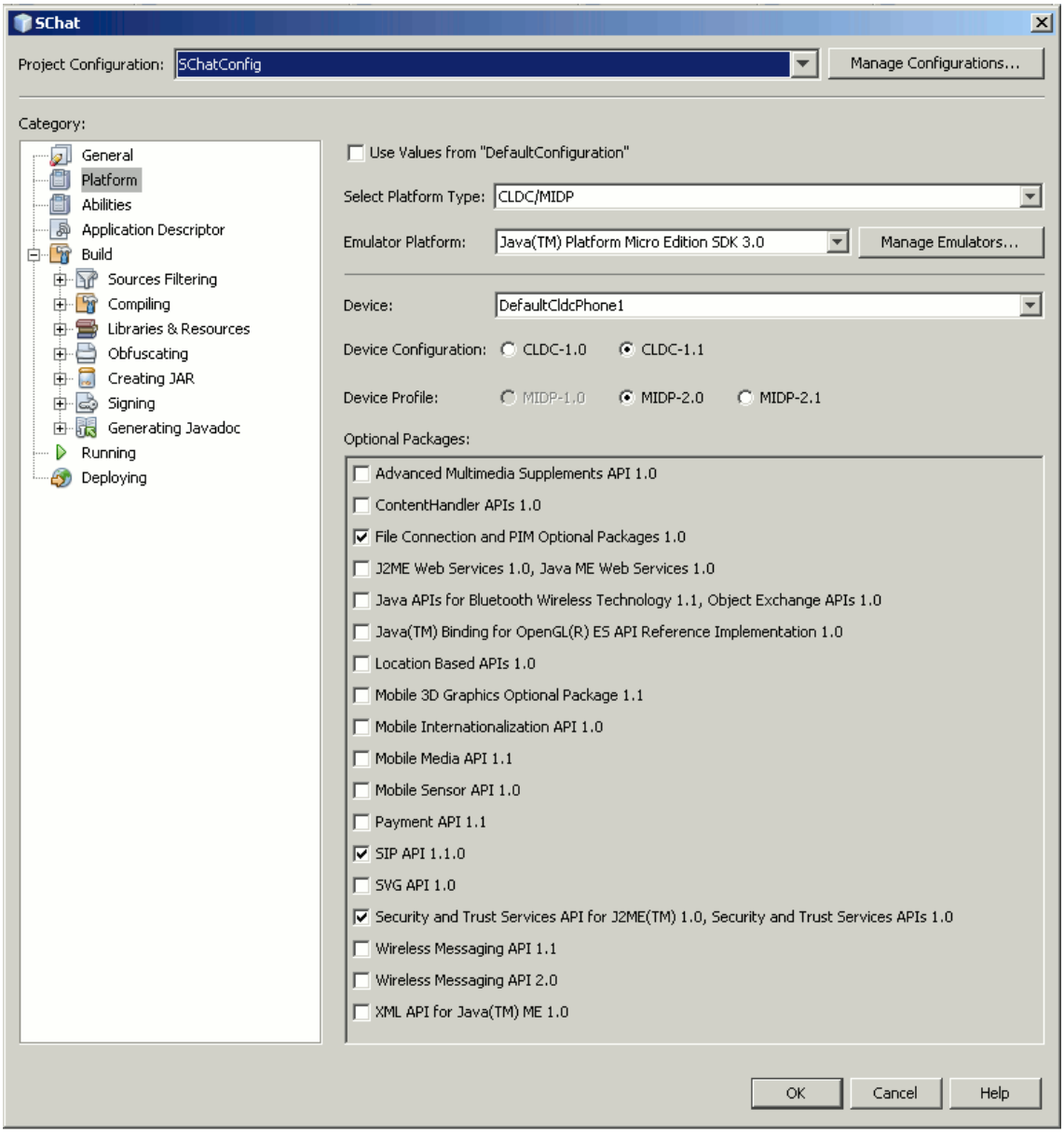


Figura 42: Configuración en simulación para el proyecto SChat

- **Entorno en terminal real:** para las pruebas en terminales reales se ha utilizado un móvil Nokia N80, el móvil cuenta con las características necesarias para la ejecución del proyecto. Las características del mismo están presentes en la Tabla 58.

Tabla 58: Características del Nokia N80

Características del terminal Nokia N80	
Developer Platform	S60 3rd Edition (Initial release)
Operating System	Symbian OS v9.1
Screen Resolution	352x416
Java Technology	JSR 139 Connected, Limited Device Configuration (CLDC) 1.1 JSR 118 MIDP 2.0 JSR 185 Java™ Technology for Wireless Industry JSR 75 FileConnection and PIM API JSR 82 Bluetooth API JSR 135 Mobile Media API JSR 172 J2ME™ Web Services Specification JSR 177 Security and Trust Services API for J2ME™ (CRYPTO and PKI packages) JSR 179 Location API for J2ME™ 1.0 JSR 180 SIP API for J2ME™ JSR 184 Mobile 3D Graphics API for J2ME™ JSR 205 Wireless Messaging API 2.0 Nokia UI API
Product ID	0x200005F9
Certificates	Symbian A, Symbian B, Symbian C, Symbian D, UTI Root
Maximum User Storage	40 MB
Free Executable RAM Memory	18 MB
NAND Memory	128 MB
SDRAM Memory	64 MB
Memory Card type	Mini SD max 2 GB
Maximum Memory Card Size	Unlimited
Maximum Jar Size	Unlimited
WLAN support	802.11b/g Nokia VoIP 2.1 WPA WPA2 (AES/TKIP)

5.2. Pruebas de simulación

Tabla 59: PSFunc01

Identificador: PSFunc01	
FUNCIONALIDAD	Crear una cuenta de usuario a partir de una dirección SIP
DESCRIPCIÓN	Se puede crear una cuenta con una dirección SIP correcta
ENTRADA	"sip:pfcdconde@ekiga.net"
RESULTADO	[Ok] la cuenta se crea correctamente
ORIGEN	RFunc01

Tabla 60: PSFunc02

Identificador: PSFunc02	
FUNCIONALIDAD	Se notifica de que no se ha podido crear la cuenta
DESCRIPCIÓN	No se puede crear una cuenta con un una dirección SIP mal formada
ENTRADA	"sip:pfcdco<>nde@ekiga.net"
RESULTADO	[Ok] la notificación se produce correctamente
ORIGEN	RVerif01

Tabla 61: PSFunc03

Identificador: PSFunc03	
FUNCIONALIDAD	Borrar una cuenta de usuario
DESCRIPCIÓN	El sistema debe borrar la cuenta, los grupos asociados a la misma y sus contactos
ENTRADA	La cuenta que se desea borrar
RESULTADO	[Ok] el proceso de borrado se produce correctamente
ORIGEN	RFunc02

Tabla 62: PSFunc04

Identificador: PSFunc04	
FUNCIONALIDAD	El sistema debe poder conectarse a una cuenta de usuario existente
DESCRIPCIÓN	El sistema envía un mensaje REGISTER correcto
ENTRADA	"sip:pfcdconde@ekiga.net"

RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark que el formato de mensaje formado a partir de la cuenta y que las transacciones de mensajes realizados en el proceso son correctos, siendo esta transacción:
	`Request: REGISTER sip:ekiga.net:5060` [Sin authorization]
	`Status: 401 Unauthorized (0 bindings)`
	`Request: REGISTER sip:ekiga.net:5060` [con authorization]
	`Status: 200 OK (1 bindings)`

ORIGEN	RFunc03

Tabla 63: PSFunc05

Identificador: PSFunc05	
FUNCIONALIDAD	El sistema debe notificar que no puede registrar una cuenta de usuario inexistente
DESCRIPCIÓN	El sistema debe identificar cuando la cuenta de usuario no existe e informar a la aplicación
ENTRADA	"sip:pfcdcondebad@ekiga.net”
RESULTADO	[Fallo] No se encuentra una solución válida para el problema identificado. El problema se origina en el funcionamiento de la librería JSR180 y el servidor de registro de ekiga.net, cuando se envía un mensaje de registro de un usuario, aunque este no exista se responde 401 Unauthorized, este mensaje se responde manera automáticamente con las credenciales establecidas lo que termina consumiendo los recursos con la consecuente excepción.
ORIGEN	RFunc03

Tabla 64: PSFunc06

Identificador: PSFunc06	
FUNCIONALIDAD	El sistema debe refrescar el mensaje de REGISTRO
DESCRIPCIÓN	Una vez establecido el registro el sistema debe refrescar el mensaje con una frecuencia menor que la cabecera EXPIRES
ENTRADA	"sip:pfcdconde@ekiga.net”
RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark que el registro se refresca periódicamente
ORIGEN	RFunc03

Tabla 65: PSFunc07

Identificador: PSFunc07	
FUNCIONALIDAD	El sistema debe notificar que no puede registrar una cuenta de usuario con un host inválido
DESCRIPCIÓN	El sistema debe identificar cuando el host no es válido e informar a la aplicación
ENTRADA	"sip:pfcdconde@ekigabad.net”
RESULTADO	[Fallo] No se encuentra una solución válida para el problema identificado. Al enviar el mensaje invocando el método asíncrono send() de SIPClientConnection se crean hilos internos de ejecución y es dentro de estos hilos donde se produce la excepción de host inválido y por ello no puede ser capturada. El listener definido en el JSR180 para la notificación de errores no recibe ninguna notificación. Por lo tanto el sistema se queda bloqueado
ORIGEN	RFunc03

Tabla 66: PSFunc08

Identificador: PSFunc08	
FUNCIONALIDAD	El sistema debe poder crear un grupo de usuarios
DESCRIPCIÓN	Tras conectarse el sistema debe crear un grupo a partir de un nombre válido
ENTRADA	“Amigos”
RESULTADO	[Ok] se comprueba que se crea el grupo de usuario
ORIGEN	RFunc04

Tabla 67: PSFunc09

Identificador: PSFunc09	
FUNCIONALIDAD	El sistema debe notificar cuando se intenta crear un grupo duplicado
DESCRIPCIÓN	Al crear un grupo el sistema debe comprobar que si ya existe el grupo de usuario y notificar a la aplicación de ello
ENTRADA	“Amigos”y “Amigos”
RESULTADO	[Ok] se comprueba que el sistema notifica a la aplicación del error
ORIGEN	RFunc04

Tabla 68: PSFunc10

Identificador: PSFunc10	
FUNCIONALIDAD	El sistema debe borrar un grupo
DESCRIPCIÓN	Al borrar un grupo se deben borrar todos los contactos en cascada
ENTRADA	“Amigos”
RESULTADO	[Ok] se comprueba que el sistema realiza el proceso de borrado
ORIGEN	RFunc05

Tabla 69: PSFunc11

Identificador: PSFunc11	
FUNCIONALIDAD	El sistema debe crear un contacto a partir de una dirección SIP
DESCRIPCIÓN	Se crea un contacto a partir de una dirección SIP válida y se asigna a algún grupo de usuario.
ENTRADA	“sip:linuxpfc@ekiga.net” y grupo “Amigos”
RESULTADO	[Ok] el contacto se crea correctamente
ORIGEN	RFunc06

Tabla 70: PSFunc12

Identificador: PSFunc12	
FUNCIONALIDAD	Se notifica de que no se ha podido crear el contacto
DESCRIPCIÓN	El sistema notifica a la aplicación cuando el contacto no tiene un formato de dirección SIP correcto
ENTRADA	“sip:linu<>xpfc@ekiga.net” y grupo “Amigos”
RESULTADO	[Ok] el sistema notifica a la aplicación
ORIGEN	RVerif01

Tabla 71: PSFunc13

Identificador: PSFunc13	
FUNCIONALIDAD	El sistema debe poder borrar un contacto
DESCRIPCIÓN	Al borrar un contacto se debe identificar a qué grupo pertenece y eliminarlo exclusivamente de ese grupo
ENTRADA	"sip:linuxpfc@ekiga.net" y grupo "Amigos2"
RESULTADO	[Ok] el contacto se elimina perfectamente
ORIGEN	RFunc07

Tabla 72: PSFunc14

Identificador: PSFunc14	
FUNCIONALIDAD	El sistema se suscribe a los cambios de estado de sus contactos
DESCRIPCIÓN	Al crear un nuevo contacto debe suscribirse a el
ENTRADA	"sip:linuxpfc@ekiga.net"

RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark que el formato de mensaje formado a partir de la cuenta y que las transacciones de mensajes realizados en el proceso son correctos, siendo esta transacción:
	`Request: SUSCRIBE sip:linuxpfc@ekiga.net:5060`
	`Status: 200 OK`
	`Request: NOTIFY [myIP]`
	`Status: 200 OK`

ORIGEN	RFunc08

Tabla 73: PSFunc15

Identificador: PSFunc15	
FUNCIONALIDAD	El sistema se suscribe a los cambios de estado de sus contactos
DESCRIPCIÓN	Al conectarse el cliente se debe suscribir a todos los contactos que ya tenga almacenados
ENTRADA	"sip:dconde@ekiga.net"

RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark que el formato de mensaje formado a partir de la cuenta y que las transacciones de mensajes realizados en el proceso son correctos, siendo esta transacción:
	`Request: SUSCRIBE sip:linuxpfc@ekiga.net:5060`
	`Status: 200 OK`
	`Request: NOTIFY [myIP]`
	`Status: 200 OK`

ORIGEN	RFunc08

Tabla 74: PSFunc16

Identificador: PSFunc16	
FUNCIONALIDAD	El sistema debe publicar el estado de una cuenta
DESCRIPCIÓN	Al conectarse el sistema debe realizar una publicación de estado indicando que éste está conectado
ENTRADA	La cuenta que se desea conectar

RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark que el formato de mensaje formado a partir de la cuenta y que las transacciones de mensajes realizados en el proceso son correctos, siendo esta transacción:
	`Request: PUBLISH sip:pfcdconde@ekiga.net:5060`
	`--Expire: 3600`
	`Status: 200 OK`

ORIGEN	RFunc09

Tabla 75: PSFunc17

Identificador: PSFunc17	
FUNCIONALIDAD	El sistema debe cancelar la publicación de estado de una cuenta
DESCRIPCIÓN	Al desconectarse el sistema debe cancelar la publicación de estado
ENTRADA	La cuenta que se desea desconectar

RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark que el formato de mensaje formado a partir de la cuenta y que las transacciones de mensajes realizados en el proceso son correctos, siendo esta transacción:
	`Request: PUBLISH sip:pfcdconde@ekiga.net:5060`
	`--Expire: 0`
	`Status: 200 OK`

ORIGEN	RFunc10

Tabla 76: PSFunc18

Identificador: PSFunc18	
FUNCIONALIDAD	El sistema debe cancelar las suscripciones de estado
DESCRIPCIÓN	Al desconectarse el sistema debe cancelar todas las suscripciones de estado
ENTRADA	Todas los contactos
RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark que se realizan todas las cancelaciones de suscripción
ORIGEN	RFunc11

Tabla 77: PSFunc19

Identificador: PSFunc19	
FUNCIONALIDAD	El sistema debe poder mandar un mensaje a un contacto
DESCRIPCIÓN	Desde la interfaz de chat se debe mandar un mensaje a un contacto
ENTRADA	“hola” al contacto “sip:linuxpfc@ekiga.net”
RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark y el cliente SIPCommunicator el correcto envío del mensaje
ORIGEN	RFunc12

Tabla 78: PSFunc20

Identificador: PSFunc20	
FUNCIONALIDAD	El sistema debe poder recibir mensaje de un contacto
DESCRIPCIÓN	Desde la interfaz de chat se debe recibir un mensaje de un contacto
ENTRADA	“que tal” desde el contacto “sip:linuxpfc@ekiga.net”
RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark y la visualización en la interfaz del mensaje que la recepción ha sido correcta
ORIGEN	RFunc13

Tabla 79: PSFunc21

Identificador: PSFunc21	
FUNCIONALIDAD	El sistema debe cancelar el registro de una cuenta
DESCRIPCIÓN	Al desconectarse el sistema debe cancelar el registro de una cuenta
ENTRADA	La cuenta que se desea desconectar

RESULTADO	[Ok] se comprueba mediante la herramienta Wireshark que el formato de mensaje formado a partir de la cuenta y que las transacciones de mensajes realizados en el proceso son correctos, siendo esta transacción:
	`Request: REGISTER sip:ekiga.net:5060` [Sin authorization]
	`--Expire: 0`
	`Status: 401 Unauthorized (0 bindings)`
	`Request: REGISTER sip:ekiga.net:5060` [con authorization]
	`--Expire:0`
	`Status: 200 OK (0 bindings)`

ORIGEN	RFunc14

5.2.1. Resumen de los resultados de simulación

Tabla 80: resumen de los resultados de las pruebas de simulación

Identificador	**Resultado**	**Identificador**	**Resultado**
PSFunc01	OK	**PSFunc02**	OK
PSFunc03	OK	**PSFunc04**	OK
PSFunc05	FALLO	**PSFunc06**	OK
PSFunc07	FALLO	**PSFunc08**	OK
PSFunc09	OK	**PSFunc10**	OK
PSFunc11	OK	**PSFunc12**	OK
PSFunc13	OK	**PSFunc14**	OK
PSFunc15	OK	**PSFunc16**	OK
PSFunc17	OK	**PSFunc18**	OK
PSFunc19	OK	**PSFunc20**	OK
PSFunc 21	OK		

5.3. Pruebas en terminal real

5.3.1. Resumen de los resultados en terminal real

En el terminal real se han realizado las mismas pruebas definidas en el entorno de simulación y conectandolo una red WIFI, los resultados se recogen en la Tabla 81.

Tabla 81: resumen de los resultados de las pruebas en terminal real

Identificador	Resultado	Identificador	Resultado
PTFunc01	OK	**PTFunc02**	OK
PTFunc03	OK	**PTFunc04**	FALLO
PTFunc05	FALLO	**PTFunc06**	OK
PTFunc07	FALLO	**PTFunc08**	OK
PTFunc09	OK	**PTFunc10**	OK
PTFunc11	OK	**PTFunc12**	OK
PTFunc13	OK	**PTFunc14**	FALLO
PTFunc15	FALLO	**PTFunc16**	OK
PTFunc17	OK	**PTFunc18**	OK
PTFunc19	OK	**PTFunc20**	FALLO
PTFunc 21	OK		

La PTFunc05 que traza contra la PSFunc05 también falla pero parece que es por distintos motivos ya que no se produce una excepción de consumo de recursos, sin embargo se obtiene un error 606 *Session Not Aceptable* pero al intentar obtener el código de error en la cabecera *Warning* no se muestra nada, por lo que aunque no se sabe directamente qué problema hay, se sospecha que tiene que ver con el acceso a la red del teléfono móvil. Para poder probar el resto de funcionalidad se ha realizado un puente en este error para que continúe como si hubiera aceptado la conexión y se ve que todos los fallos en el terminal real ocurren en los casos en que otros agentes de usuario intentan enviar información al terminal. Otras pruebas realizadas:

- Con un terminal N95 y la misma red WIFI se obtienen los mismos resultados.
- Con una red GPRS en el terminal N80, los resultados han sido peores obteniendo los siguientes errores de symbian -17702 y -17770 estos errores indican petición SIP pendiente y KErrSipCodecURI respectivamente y al realizar el puente para aceptar la conexión se comprueba que no hay funcionalidad.
- Con el terminal N80 y una red WIFI que asigna IP pública se siguen obteniendo los mismos resultados de Session Not Aceptable y error -17770, en este caso en algunas ocasiones aparece también el código de error -18 KErrNotReady que indica que el terminal no puede realizar operaciones I/O en ese momento, aparentemente este error se produce ante las pérdidas de conectividad que hay con la red WIFI seleccionada.

Capítulo 6.

Historia del proyecto

El proyecto se inicia como una línea de investigación de la tecnología SIMPLE en terminales móviles utilizando tecnología Java. El proyecto está encabezado por la Universidad Carlos III de Madrid comenzando la investigación en enero de 2010 y terminando en septiembre del mismo año.

Al contar el proyecto con otras tareas, se ha realizado un registro de las horas dedicadas a esta línea de investigación, a partir de este registro se describirán las distintas etapas realizadas.

Posteriormente se reflejarán algunas dificultades y problemas encontrados durante la realización del proyecto y que han supuesto un retraso importante en el desarrollo del mismo o un impedimento para completar algunos requisitos de partida.

6.1. Distribución temporal del proyecto

Se describen las distintas etapas de realización del proyecto:

- **Lectura documentación:** se realiza una lectura de los RFCs del protocolo SIMPLE y de SIP, se estudian las tecnologías Java que se pueden utilizar determinando que se iniciará primero con la línea de Java Card versión 3(JC3)(43), este periodo tiene una duración aproximada del 17% de un mes[4].

  JC3 es una tecnología desarrollada por Sun Microsystems que ofrece un entorno para la implementación de aplicaciones destinadas a *smart cards* la principal ventaja de esta tecnología es la portabilidad que ofrece.

- **Línea de investigación con JC3:** para la realización del proyecto con esta tecnología sería necesario implementar una librería SIP y otra librería para SIMPLE. Se estudian las conexiones UDP y se comprueba que pese a que la especificación indica que son posibles, el kit de desarrollo que se dispone no las implementa por lo que no se puede realizar la librería SIP. Se decide retomar la investigación utilizando la tecnología J2ME.
- **Línea de investigación con J2ME:** el estudio de esta tecnología revela las especificaciones JSR 180 para SIP y JSR 164-165 para SIMPLE, además se dispone de una implementación del JSR 180 para terminales Nokia. Tras realizar algunas pruebas sobre las conexiones SIP se determina que aunque se existen algunas restricciones en cuanto a la creación de servidores en modo dedicado se puede utilizar el modo compartido, por lo que la implementación parece posible.
- **Implementación del proyecto:** el desarrollo se realiza durante un periodo aproximado de mes y medio[4], la implementación de la librería SIMPLE de la que se dispone una especificación mientras se realiza el análisis y diseño del prototipo. Hacia la mitad del desarrollo de la librería se comienza el desarrollo del prototipo que permite ir realizando más pruebas sobre la librería.
- **Pruebas:** las pruebas finales duran un periodo aproximado del 20% de un mes[4], se inician con la librería desarrollada al 90% el prototipo está muy avanzado, pero se descubre un problema en el manejo de transacciones lo que implica un cambio significativo en la librería.
- **Realización de la memoria:** la realización de la memoria se lleva a cabo en un periodo aproximado de un mes[4] mientras se realizaba esta tarea se han seguido realizando pruebas al proyecto e intentado realizar mejoras que permitieran soslayar el problema encontrado en los terminales reales.

[4] Un mes se contabiliza como en el presupuesto correspondiendo a 131,25 horas.

6.2. Dificultades y soluciones adoptadas

El proyecto ha tenido una importante reimplementación de la estructura del código en cuanto al tratamiento de las transacciones; en una primera instancia las conexiones se controlaban a través de los *listener* ofrecidos por la librería de SIP. La utilización de *listener* en las peticiones generaba la necesidad de identificar el origen de las transacciones, según la especificación del JSR180 existe restricción en el acceso a determinadas cabeceras, sin embargo se debería poder utilizar la cabecera `"Call-ID"` con este objetivo puesto, que es consistente en la transacción completa y su acceso debería estar restringido en escritura pero no en lectura. Este modelo se implementó teniendo resultados satisfactorios en los test de simulación, pero la implementación real que utilizan los móviles de Nokia no devolvía ningún valor al intentar leer la cabecera `"Call-ID"`.

La solución final adoptada supuso una reestructuración en el manejo de transacciones que elimina el *listener* para las peticiones ya que en el modelo actual se crea un hilo auto gestionado para cada una. Esto simplifica la utilización de los *timers* y mejora el mantenimiento de recursos, evitando tener que identificar las transacciones lo que elimina la necesidad de leer la cabecera `"Call-ID"`.

El segundo problema encontrado no se ha podido solucionar, éste consiste en que en los terminales reales en los cuales se ha comprobado que en el registro de la aplicación se obtiene un mensaje `606 Session Not Acceptable`, la lectura de la cabecera `Warning` no aporta ninguna información adicional por lo que no se pude identificar correctamente por qué no se acepta la sesión. Examinando las trazas de mensaje del móvil y las trazas de simulación se observa la siguiente diferencia en las cabeceras `Via`:

```
Via: SIP/2.0/UDP 163.117.141.223:5060
```

```
Via: SIP/2.0/UDP
10.1.5.82:5060;received=163.117.139.108:5060;rport=5060
```

La última observación al problema reside en que ante cada transacción que se desea realizar el terminal móvil pide que se active la red *wifi* y esta no se mantiene constantemente por lo que se sospecha que el terminal no es capaz de mantener el servidor para la recepción de conexiones SIP.

Capítulo 7.

Presupuesto

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DEL PROYECTO

Autor
Daniel Conde García
Departamento
Departamento de telemática
Descripción del proyecto
Titulo Implementación de una librería para el protocolo SIMPLE en J2ME
Duración 3,83 meses
Tasa de costes indirectos 20%
Presupuesto total del proyecto
Cantidad en euros 13.115 €
Desglose presupuestario

PERSONAL

Firma	Apellidos, Nombre	NIF	Categoría	Dedicación [a)]	Coste (ho/m)	Coste Imputable
	Conde García, Daniel	00000000A	Ingeniero	3,50	2.694,39 €	9.430,37 €
	Campo Vázquez, Mª Celeste	00000000A	Ingeniero Senior	0,33	4.289,54 €	1.415,55 €
			Ho/m	3,83	**Total**	**10.845,91 €**

a) 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1.575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste	% Uso	Dedicación meses	Depreciación	Coste Imputable
Portátil Aspire	900 €	100%	0,4	60	6,00 €
Portátil Apple	1.500 €	100%	0,1	60	2,50 €
PC de desarrollo	700 €	100%	3,0	60	35,00 €
PC de pruebas	700 €	100%	0,8	60	9,33 €
Nokia N80	50 €	100%	0,5	60	0,42 €
				Total	**53,25 €**

COSTES DIRECTOS

Descripción	Empresa	Coste Imputable
Material de oficina	Consumibles-X S.A.	30,00 €
	Total	**30,00 €**

Resumen de costes totales

Personal	10.846 €
Amortizaciones	53 €
Costes directos	30 €
Costes indirectos	2.186 €
TOTAL	***13.115 €***

El presupuesto total de este proyecto asciende a la cantidad de **13.115 Euros**.

Leganés a 07 de septiembre de 2010

Fdo. Daniel Conde García

Capítulo 8.

Conclusiones y trabajos futuros

8.1. Conclusiones

Bajo el enfoque de los objetivos del proyecto se puede decir que si bien no se han alcanzado todos, si se han alcanzado la gran mayoría de ellos; los objetivos de implementación de la librería y del prototipo se han cumplido satisfactoriamente.

Dentro de la evaluación, el escenario de simulación ofrece unos buenos resultados frente al escenario de terminales reales donde no se han alcanzado los objetivos; sin embargo algunas pruebas realizadas en el segundo escenario puenteando el registro arroja algo de luz ya que el terminal es capaz de enviar datos correctos a otros agentes de usuario. Por lo que es la recepción de datos la que no está realizando su función probablemente por un problema de configuración de la red en el terminal.

Bajo el enfoque de proyecto final de carrera, este proyecto ha supuesto la oportunidad de demostrar los conocimientos adquiridos en Ingeniería Informática durante estos años, los cuales abarcan tanto la capacidad de realizar un proceso de análisis y diseño para un software como la capacidad de implementar dicho software; incluyendo una buena predisposición para adaptarse y aprender nuevas tecnologías en un periodo corto de tiempo.

8.2. Trabajos Futuros

Debido a las limitaciones que se han de definir en la realización de un proyecto de fin de carrera, han quedado fuera del alcance de éste un conjunto de funcionalidades que serían interesantes investigar e implementar:

- **Funcionalidad en terminales reales:** como se ha visto en los puntos 5.3 y 6.2 actualmente existe un problema en la ejecución en terminales reales. Sería imprescindible para continuar con el desarrollo ahondar en el tema y conseguir una versión funcional.
- **Desarrollar un chat real:** con el prototipo se ha visto la plausibilidad de la tecnología, por lo que se podría hacer una explotación real en los terminales.
- **Implementar otro tipo de transferencia:** como SIMPLE se define sobre SIP se dispone de toda la funcionalidad de este último para la transferencia de otros tipos de datos. Por ello se podría ampliar la librería para que aceptara imágenes, voz e incluso vídeo.
- **Ampliar la seguridad:** actualmente el protocolo se implementa sobre conexiones SIP (sip), pero tendría que estudiarse la dificultad de realizar una implementación con *secure* SIP (sips).
- **Implementar las librerías con JC3:** las ventajas en cuanto a portabilidad que se obtendrían al implementar la librería en esta tecnología serían enormes, como se vio en el punto 6.1 se encontró un problema con las conexiones UDP por lo que se abandonó esta línea de investigación; sin embargo sería muy interesante retomar esta línea cuando la tecnología madure un poco.

Glosario, acrónimos y siglas

AES Advanced Encryption Standard, estándar de cifrado de bloque

API Application Programming Interface, conjunto de funciones y procedimientos que ofrece una librería

BBS Bulletin Board System

Chat Término de origen inglés que designa una comunicación escrita realizada de manera instantánea a través de Internet entre dos o más personas

CLDC Connected Limited Device Configuration, framework de desarrollo de aplicaciones J2ME

CTSS Compatible Time-Sharing System, SO de tiempo compartido

DCC Direct Client to Client

ESA European Space Agency

Firewall Es un equipo de hardware o software utilizado en las redes de ordenadores para prevenir algunos tipos de comunicaciones prohibidos por la política de red

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

IANA Internet Assigned Numbers Authority

IDE Integrated Development Environment

IETF Internet Engineering Task Force

IRC Internet Relay Chat

J2ME Java 2 Micro Edition, versión de Java destinada a pequeños dispositivos

JSR	Java Specification Request
Metodología	Marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información
MIDlet	Programa en lenguaje de programación Java para dispositivos embedidos
MIT	Massachusetts Institute of Technology
Multics	Multiplexed Information and Computing Service, SO de tiempo compartido
NAT	Network Address Translation
OSCAR	Open System for CommunicAtion in Realtime
Pizarra electrónica	Sistema de software que simula una pizarra tradicional y permite a los usuarios interactuar mediante gráficos
RC4	Rivest Cipher 4, estándar de cifrado de flujo
RFC	Request For Comments
SIMPLE	Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
TCP	Transport Control Protocol
TLS	Transport Layer Security
TOC2	Talk to OSCAR protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language, lenguaje de modelado de ingeniería del software
URI	Uniform Resource Identifier
VoIP	Conjunto de protocolos que permiten establecer una comunicación de transporte de información principalmente de Voz en tiempo real.
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
YMSG	Yahoo! Messenger Protocol
Singleton	Patrón de diseño que limita la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto

Referencias

1. **ESA BSSC.** Guide to applying the ESA software engineering standards to small software projects. [En línea] Mayo de 1996. http://emits.esa.int/emits-doc/e_support/Bssc962.pdf. Última consulta: Agosto 2010

2. **Vleck, Tom Van.** The History of Electronic Mail. [En línea] 25 de Mayo de 2008. http://www.multicians.org/thvv/mail-history.html. Última consulta: Agosto 2010

3. **Wikipedia, the free encyclopedia.** talk (software) - Wikipedia, the free encyclopedia. [En línea] 17 de Agosto de 2010. http://en.wikipedia.org/wiki/Talk_(software). Última consulta: Agosto 2010

4. —. Talker - Wikipedia, the free encyclopedia. [En línea] 04 de Julio de 2010. http://en.wikipedia.org/wiki/Talker. Última consulta: Agosto 2010

5. **Jiang, Renée.** Background. [En línea] 2006. http://www.ocf.berkeley.edu/~yrjiang/background.htm. Última consulta: Agosto 2010

6. **The New York Times Company.** A "Quantum" Leap for AIM. [En línea] 2010. http://im.about.com/od/imbasics/a/imhistory_3.htm. Última consulta: Agosto 2010

7. **ICQ Inc.** About the ICQ - ICQ.com. [En línea] 1998-2007. http://www.icq.com/info/icqstory.html. Última consulta: Agosto 2010

8. **IBM.** IBM Design - Design gallery: Software. [En línea] http://www-01.ibm.com/software/ucd/gallery/sametime_software.html. Última consulta: Agosto 2010

9. **SIMPLE IETF Group.** SIP for Instant Messaging and Presence Leveraging Extensions (simple) - Charter. [En línea] http://datatracker.ietf.org/wg/simple/charter/. Última consulta: Junio 2010

10. **The XMPP Standards Foundation.** History - The XMPP Standards Foundation. [En línea] 27 de Enero de 2010. http://xmpp.org/about-xmpp/history/. Última consulta: Agosto 2010

11. **United States Patent and Trademark Office.** SUMMARY OF FINAL DECISIONS ISSUED BY THE TRADEMARK TRIAL AND APPEAL BOARD. [En línea] 16 de Enero de 2006. http://www.uspto.gov/web/offices/com/sol/foia/ttab/decsum/2006/16jan06.pdf. Última consulta: Agosto 2010

12. **The Internet Engineering Task Force (IETF).** RFC 2810 - Internet Relay Chat: Architecture. [En línea] Abril de 2000. http://tools.ietf.org/html/rfc2810. Última consulta: Agosto 2010

13. —. RFC 2811 - Internet Relay Chat: Channel Management. [En línea] Abril de 2000. http://tools.ietf.org/html/rfc2811. Última consulta: Agosto 2010

14. —. RFC 2812 - Internet Relay Chat: Client Protocol. [En línea] Abril de 2000. http://tools.ietf.org/html/rfc2812. Última consulta: Agosto 2010

15. —. RFC 2813 - Internet Relay Chat: Server Protocol. [En línea] Abril de 2000. http://tools.ietf.org/html/rfc2813. Última consulta: Agosto 2010

16. **mIRC Co. Ltd.** mIRC: Internet Relay Chat client. [En línea] http://www.mirc.com/. Última consulta: Agosto 2010

17. **Zelezny, Peter.** XChat: Multiplatform Chat Program. [En línea] 28 de Agosto de 2010. http://xchat.org/. Última consulta: Agosto 2010

18. **The Internet Engineering Task Force (IETF).** Instant Messaging / Presence Protocol Requirements. [En línea] Febrero de 2000. http://www.ietf.org/rfc/rfc2779.txt. Última consulta: Agosto 2010

19. **Pidgin.** Pidgin, the universal chat client. [En línea] http://www.pidgin.im/.

20. **Cerulean Studios.** Trillian - IM, Astra, Windows Live, Facebook, Twitter, Yahoo, MySpace, AIM, Email, and more! [En línea] http://www.trillian.im/.

21. **AOL.** OSCAR Protocol | dev.aol.com. [En línea] Marzo de 2008. http://web.archive.org/web/20080308233204/http://dev.aol.com/aim/oscar/. Última consulta: Agosto 2010

22. *An Analysis of the Skype Peer-to-Peer Internet Telephony.* **Schulzrinne, Salman A. Baset and Henning.** New York NY 10027 : Department of Computer Science, Columbia University, September 15, 2004.

23. **America Online, Inc.** Version: TOC1.0. [En línea] http://terraim.cvs.sourceforge.net/viewvc/terraim/terraim/src/toc/. Última consulta: Agosto 2010

24. **Wikipedia, the free encyclopedia.** Yahoo! Messenger Protocol - Wikipedia, the free encyclopedia. [En línea] 26 de Agosto de 2010. http://en.wikipedia.org/wiki/YMSG. Última consulta: Agosto 2010

25. —. Zephyr (protocol) - Wikipedia, the free encyclopedia. [En línea] 07 de Juli de 2010. http://en.wikipedia.org/wiki/Zephyr_(protocol). Última consulta: Agosto 2010

26. **The Internet Engineering Task Force (IETF).** SIP: Session Initiation Protocol. [En línea] Junio de 2002. http://www.ietf.org/rfc/rfc3261.txt. Última consulta: Agosto 2010

27. —. Session Initiation Protocol (SIP)-Specific Event Notification. [En línea] Junio de 2002. http://www.ietf.org/rfc/rfc3265.txt. Última consulta: Agosto 2010

28. —. A Presence Event Package for the Session Initiation Protocol (SIP). [En línea] Agosto de 2004. http://www.ietf.org/rfc/rfc3856.txt. Última consulta: Agosto 2010

29. —. Session Initiation Protocol (SIP) Extension for Event State Publication. [En línea] Octubre de 2004. http://www.ietf.org/rfc/rfc3903.txt. Última consulta: Agosto 2010

30. —. Presence Information Data Format (PIDF). [En línea] Agosto de 2004. http://www.ietf.org/rfc/rfc3863.txt. Última consulta: Agosto 2010

31. —. Session Initiation Protocol (SIP) Extension for Instant Messaging. [En línea] Diciembre de 2002. http://www.ietf.org/rfc/rfc3428.txt. Última consulta: Agosto 2010

32. **Sun Microsystems.** Java ME Technology. [En línea] http://www.oracle.com/technetwork/java/javame/tech/index.html.

33. **Process, Java Community.** *Mobile Service Architecture 2 Specification Public Review Version 0.19.* 2008.

34. **Java Community Process.** The Java Community Process(SM) Program. [En línea] http://www.jcp.org/en/home/index.

35. **Sun Microsystems.** FAQ. *The Java Community Process.* [En línea] http://www.jcp.org/en/introduction/faq. Última consulta: Agosto 2010

36. **Java Community Process.** JSR-000164 SIMPLE Presence 1.0 - Final Release. [En línea] 2010. http://jcp.org/aboutJava/communityprocess/final/jsr164/index.html.

37. —. JSR-000165 SIMPLE Instant Messaging 1.0 - Final Release. [En línea] 2010. http://jcp.org/aboutJava/communityprocess/final/jsr165/index.html.

38. —. JSR 180: SIP API for J2METM. *The Java Community Process (SM) Program.* [En línea] http://jcp.org/en/jsr/detail?id=180.

39. **Object Management Group.** Object Management Group - UML. *Object Management Group - UML.* [En línea] 2009. http://www.uml.org/.

40. **The Internet Engineering Task Force (IETF).** Key words for use in RFCs to Indicate Requirement Levels. [En línea] Marzo de 1997. http://www.ietf.org/rfc/rfc2119.txt.

41. **TrekBuddy.** Devices. [En línea] 2010. http://wiki.trekbuddy.net/index.php/Devices. Última consulta: Septiembre 2010

42. **Nokia.** Forum Nokia - Your idea. Our tools. [En línea] 2010. http://www.forum.nokia.com/.

43. **Sun Microsystems.** Java Card versión 3. [En línea] http://www.oracle.com/technetwork/java/javacard/overview/index.html.

44. **Perea, Rogelio Martinez.** *Internet Multimedia Communications Using SIP: A Modern Approach Including Java® Practice.* s.l. : Morgan Kaufmann, January 15, 2008. ISBN-10: 0-12-374300-1, ISBN-13: 978-0-12-374300-8.

45. **Nokia.** Forum Nokia - Java API Specifications - Java - Specifications. *JSR 180.* [En línea] http://www.forum.nokia.com/Develop/Java/Documentation/Java_API_specifications.xhtml#jsr180. Última consulta: Agosto 2010

Anexo I: Guía de configuración del entorno de desarrollo

Requisitos:

Se debe tener instalado el siguiente software para configurar el proyecto y poder seguir su desarrollo:

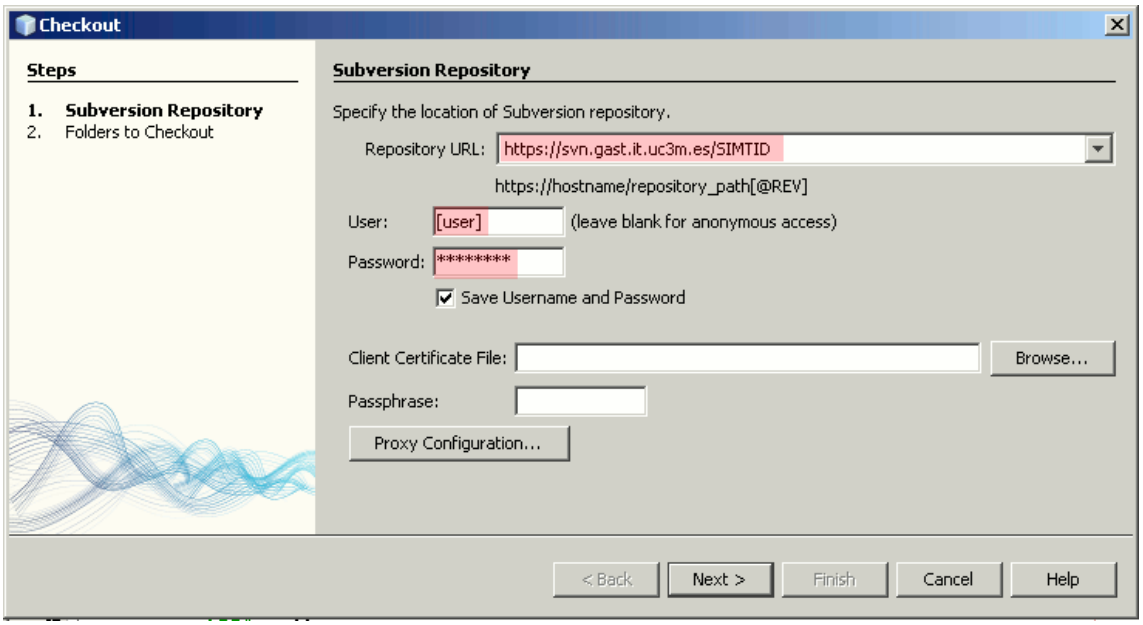
- NetBeans IDE 6.8 o superior[5]
- Java(TM) Platform Micro Edition SDK 2.5.2 o superior[6]
- Plugin de NetBeans para la utilización de repositorios SVN

[5] http://netbeans.org/
[6] http://www.oracle.com/technetwork/java/download-135801.html

Descarga del proyecto:

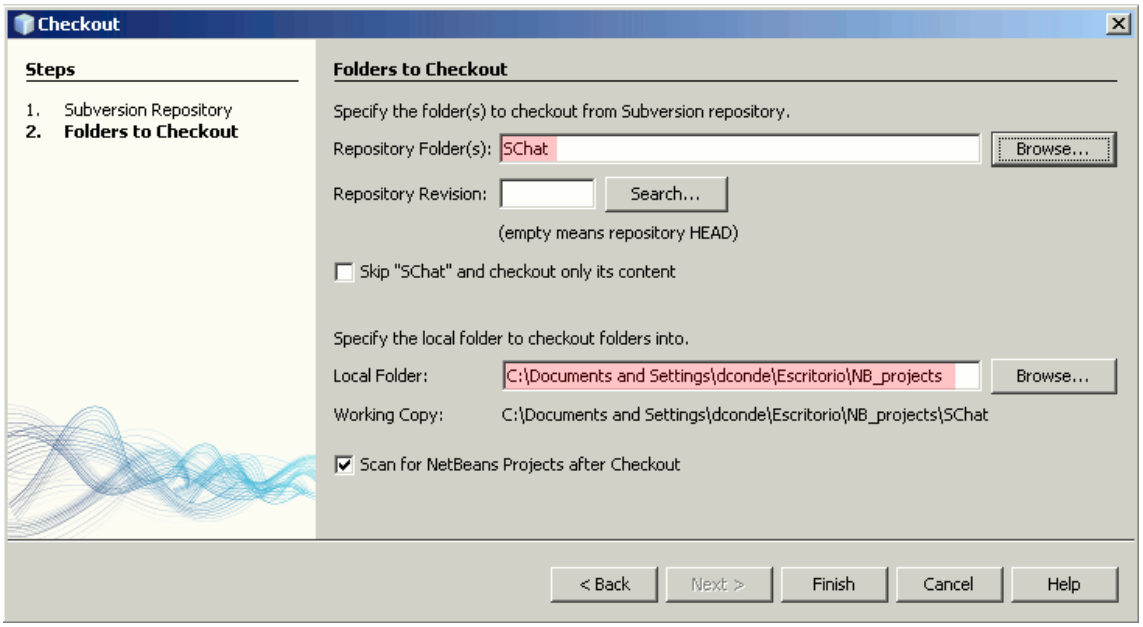
Para obtener las fuentes del proyecto y poder continuar su desarrollo hay que dirigirse al menú `Team/Subversion/Checout…` y configurar los siguientes:

- **Repository URL:** https://svn.gast.it.uc3m.es/SIMTID
- **User:** el usuario de acceso al repositorio
- **Password:** el password asociado a ese usuario



Se presiona `Next` y rellenan los siguientes campos presionando `Finish` al terminar:

- **Repository Folder:** SChat
- **Local folder:** con la dirección local en la que se desea almacenar



Configuración:

Para la configuración del proyecto realizamos botón derecho en el proyecto y seleccionamos `properties` apareciendo la siguiente pantalla:

- **Select Platform Type:** CLDC/MIDP
- **Emulator Platform:** Java™ Platform Micro Edition SDK 3.0
- **Device Configuration:** CLCD-1.1
- **Device Profile:** MIDP-2.0
- **Optional Packages:** File Connection, SIP API 1.1, Security and Trust Services